Files from C:\RON\AUTOSPLT\INTENT\INTENT.ZIP

DRL_CMDS.C96	08/29/94
PR_S_I_S.C96	06/30/94
SEL_GEAR.C96	08/19/94
SEQ_SHFT.C96	08/04/94
TRNS_ACT.C96	08/18/94

```
/*****************************************************************************
 *
 *        Unpublished and confidential.  Not to be reproduced,
 *        disseminated, transferred or used without the prior
 *        written consent of Eaton Corporation.
 *
 *        Copyright Eaton Corporation, 1993-94.
 *        All rights reserved.
 *
 *****************************************************************************
 *
 * Filename: [redacted]      (AutoSplit)
 *
 * Description:
 *      The functions in this file will perform the required operations
 *      for controlling driveline components on the J1939 communication link.
 *
 * Part Number:  <none>
 *
 * $Log:   R:\ashift\vcs\drl_cmds.c9v  $
 *
 *    Rev 1.9   25 Feb 1994 16:07:40   schroeder
 * Removed predip's (experimental) BIN 8 ramp off rate--no longer needed.
 * Added condition to predip's torque bumps: timers are frozen until
 * actual_engine_pct_trq responds.  Important when engine brakes are on.
 * Made RECOVERY_STEP_TABLE[] a constant value--all ratios used 8%.
 *
 *    Rev 1.8   21 Feb 1994 15:07:26   schroeder
 * Replaced shiftability_mode with new flag, engine_brake_available.
 *
 *    Rev 1.7   03 Feb 1994 15:57:28   schroeder
 * Added setting of command_ETC1 to each command function, enabling
 * overspeed during control_engine_predip() and control_engine_sync() and
 * disabling overspeed during all others.
 *
 *    Rev 1.6   17 Nov 1993 09:50:10   amsallen
 * In the initiate function, net_engine_pct_trq was replaced with
 * act_engine_pct_trq since desired_engine_pct_trq values are in indicated
 * power not net_power. Also the sync_timer timeout was reduced to 100 for
 * down shifts and 200 for upshifts(1 second and 2 seconds) rather than 200
 * for all shifts. Also the delay at 0 torque after a time out was increase
 * from 150msec to 250 mesec.
 *
 *    Rev 1.5   04 Nov 1993 09:16:08   schroeder
 * In RECOVERY_STEP_TABLE[], restored the value for sixth gear to 8%.
 *
 *    Rev 1.4   02 Nov 1993 09:40:06   schroeder
 * Replaced demand_engine_pct_trq with pct_demand_at_cur_sp.  Engine
 * speed limit during torque limit operation changed from high idle to
 * 8031 rpm, as suggested in J1939/71.
 *
 *    Rev 1.3   11 Oct 1993 14:22:40   schroeder
 * Removed cruise_control_active flag; replaced accel pedal with
 * demand_engine_pct_trq.
 *
 *    Rev 1.2   22 Sep 1993 10:48:22   amsallen
 * The function control engine sync was changed to resolve OR 3235ma08.deb,
 * clunky low throttle high range shifts. The engine target now moves above
 * sync when input_speed - sync < 40 rpm and transmission position = engaging
 * rather than just tp = engaging. See OR 3235ma08.deb for additional details.
 *
 *    Rev 1.1   01 Sep 1993 14:09:52   schroeder
 * In control_engine_sync, modified dither to insure that the engine
 * target stays below sync for a range shift.  Also, the dither amount
 * increases from 35rpm to 100rpm, then repeats.
 *
 *    Rev 1.0   29 Jul 1993 16:40:40   schroeder
 * Initial revision.
 *
 *****************************************************************************/


/*****************************************************************************
 *
 * Header files included.
 *
 *****************************************************************************/

#include <exec.h>           /* executive information */
```

```
#include <c_regs.h>         /* KB internal register definitions */
#include <vwslib.h>         /* contains common global defines */
#include "cont_sys.h"       /* control system information */
#include "conj1939.h"       /* defines interface to j1939 control module */
#include "drl_cmds.h"       /* driveline commands information */
#include "trn_tbl.h"        /* transmission table data structures */
#include "sel_gear.h"       /* access to speed filter values */
#include "calc_spd.h"
#include "trns_act.h"

#pragma noreentrant

/*****************************************************************************
 *
 * #defines local to this file.
 *
 *****************************************************************************/

#define US_PER_LOOP 10000U

#define ACTIVE_RECOVERY_GEAR 10   /* rule out boosting downs for now */

/*****************************************************************************
 *
 * Constants and variables declared by this file.
 *
 *****************************************************************************/

/* public */

register uchar engine_commands;
register uchar engine_status;
uchar desired_sync_test_mode;
/* uint  desired_eng_spd_new;    */
/* uint  desired_eng_spd_delta;  */
/* uint  desired_eng_spd_diff;   */
uint  desired_engine_speed_test;
uint  desired_engine_speed_ramp;
uchar desired_engine_speed_timer;
uchar desired_engine_speed_time;
uchar eng_brake_command;
uchar eng_brake_assist;
uchar positive_pedal_trans;
uchar sync_first_pass_timer;
uchar clutch_state;
uint  clutch_slip_speed;
int   dos_filtered;
int   overall_error;
unsigned int  os_based_on_rcs;
unsigned int  input_speed_filtered;
unsigned long is_filtered_bin8;
unsigned int  output_speed_filtered;
unsigned long os_filtered_bin8;
signed   int  input_speed_accel_filtered;
signed   long dis_filtered_bin8;

char  eng_percent_torque_filtered;
char  percent_torque_accessories;
char  needed_percent_for_zero_flywheel_trq;
uchar zero_flywheel_trq_timer;
uchar zero_flywheel_trq_time;
uchar accelerator_pedal_position_old;
int   input_shaft_accel_calculated;
uint  gos;          /* overall destination gear ratio * output speed BIN 0 */
int   gos_signed;   /* overall destination gear ratio * output speed BIN 0 */

uint  gos_current_gear;  /* overall current gear ratio * output speed BIN 0 */

unsigned char  sync_first_pass;
unsigned int   sync_maintain_timer;
  signed int   sync_offset;
  signed int   sync_dos_offset;
  signed int   sync_dos_offset_K1;
  signed int   sync_speed_modified;
unsigned char  intent_to_shift;
         char  intent_final_pct_trq;
         char  intent_ramp_off_rate;

/* local */
```

```
static uint predip_timer_1;
static uchar predip_timer_2;
static uchar predip_timer_3;
static char predip_torque_bump_value;
static uchar predip_torque_bump_time;

static uint sync_on_timer;
static uint sync_off_timer;
static uchar sync_dither_timer;

static uint torque_limit;
static uchar recovery_cancel_timer;
static uint recov_coast_down_tmp1;
static uint recov_coast_down_tmp2;

static int dgos;
static int lpf_output_accel;


#pragma EJECT
```

```
/*****************************************************************************
 *
 *                          PREDIP MODE CONSTANTS
 *
 *****************************************************************************/

#define PREDIP_ZERO_FDBK_TIME          40        /* 0.40s @10ms period */
#define PREDIP_TORQUE_ZERO_TIME        60        /* 0.60s @10ms period */
#define PREDIP_NORMAL_TIME            200        /* 2.00s @10ms period */
#define TORQUE_RAMP_OFF_RATE            1        /* 1% (per loop) */

#define PREDIP_TORQ_BUMP_VALUE_LO       0        /* 0% */
#define PREDIP_TORQ_BUMP_TIME_LO       15        /* 0.15s @10ms period */

#define PREDIP_TORQ_BUMP_VALUE_MED      5        /*  5% */
#define PREDIP_TORQ_BUMP_TIME_MED      25        /* 0.25s @10ms period */

#define PREDIP_TORQ_BUMP_VALUE_HI      10        /* 10% */
#define PREDIP_TORQ_BUMP_TIME_HI       30        /* 0.30s @10ms period */

/*****************************************************************************
 *
 *                          SYNC MODE CONSTANTS
 *
 *****************************************************************************/

#define SYNC_DITHER_TIME_ABOVE         20        /* 0.20s @10ms period */
#define SYNC_DITHER_TIME_BELOW         30        /* 0.30s @10ms period */
#define SYNC_DITHER_RPM                35        /* 35 rpm             */
#define SYNC_DITHER_FIRST_TIME        255        /* DUMMY VALUE        */

#define  MAINTAIN_SYNC_TIME           500        /* 5.00 Sec           */
#define  SYNC_FIRST_PASS_TIME         250        /* 2.50 Sec           */
#define  THREE_PERCENT                  3
#define  ENG_RESPONSE_UPSHF_TIME       10        /* 10 msec            */
#define  ENG_RESPONSE_DNSHF_TIME       10        /* 10 msec            */
#define  SYNC_DOS_OFFSET_CONSTANT    2816        /* 11 BIN 8           */

/*****************************************************************************
 *
 *                        RECOVERY MODE CONSTANTS
 *
 *****************************************************************************/

#define RECOVERY_CANCEL_TIME           10        /* 0.10s @10ms period */
#define RECOVERY_CANCEL_OFFSET         20        /* 20% BIN 0 */
#define RECOVERY_TORQUE_STEP         1280        /* 5% BIN 8 */
#define THLO_DS_ENG_DECAY_K1          450
#define THLO_DS_ENG_DECAY_RAMP          1        /* 1 rpm BIN 0 */
#define THLO_DS_FINISHED_DELTA        200        /* 200 rpm BIN 0 */

static const uint RECOVERY_RATE_TABLE[23] =
{
   0,          /* -4 */
   0,          /* -3 */
   128,        /* -2 : 0.50% per loop BIN 8 */
   128,        /* -1 : 0.50% per loop BIN 8 */
   128,        /*  0 : 0.50% per loop BIN 8 */
   128,        /*  1 : 0.50% per loop BIN 8 */
   128,        /*  2 : 0.50% per loop BIN 8 */
   128,        /*  3 : 0.50% per loop BIN 8 */
   128,        /*  4 : 0.50% per loop BIN 8 */
   192,        /*  5 : 0.75% per loop BIN 8 */
   192,        /*  6 : 0.75% per loop BIN 8 */
   192,        /*  7 : 0.75% per loop BIN 8 */
   281,        /*  8 : 1.10% per loop BIN 8 */
   281,        /*  9 : 1.10% per loop BIN 8 */
   281,        /* 10 : 1.10% per loop BIN 8 */
   0,          /* 11 */
   0,          /* 12 */
   0,          /* 13 */
   0,          /* 14 */
   0,          /* 15 */
   0,          /* 16 */
   0,          /* 17 */
   0           /* 18 */
};
```

```
#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  initialize_driveline_data
 *
 * Description:
 *      This function, called after all resets, will initialize the system
 *      copy of driveline related data received from the communications link.
 *
 *****************************************************************************/

static void initialize_driveline_data(void)
{
   accelerator_pedal_position = 0;
   engine_communication_active = FALSE;
   engine_brake_available = FALSE;
   eng_brake_command = ENG_BRAKE_IDLE;   /* should init with engine_commands */
   clutch_state = ENGAGED;
   positive_pedal_trans = FALSE;
   zero_flywheel_trq_timer = 0;
   zero_flywheel_trq_time = 0;
   percent_torque_accessories = 3;

   desired_sync_test_mode = FALSE;   /* debug use only - delete later */
   desired_engine_speed_test = 0 ;   /* debug use only - delete later */
   desired_engine_speed_ramp = 0 ;   /* debug use only - delete later */
   desired_engine_speed = 0;
  /* desired_eng_spd_new = 0;       */
  /* desired_eng_spd_diff = 0;      */
  /*  desired_eng_spd_delta = 10;   */
   desired_engine_speed_timer = 0;
   sync_dos_offset_K1 = SYNC_DOS_OFFSET_CONSTANT;
   desired_engine_speed_time = 0;
   intent_to_shift = FALSE;
   intent_final_pct_trq = 0;
   intent_ramp_off_rate = 1;

}


#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_predip
 *
 * Description:
 *      Determines throttle command for predip mode.
 *
 *      After a reasonable delay for the transmission to pull to neutral the
 *      torque will be cycled from zero to a determined value to help the
 *      transmission achieve neutral.
 *
 *****************************************************************************/

static void control_engine_predip(void)
{
   if (engine_status != ENGINE_PREDIP_MODE)
   {
      engine_status = ENGINE_PREDIP_MODE;

      predip_timer_1 = 0;
      predip_timer_2 = 0;
      predip_timer_3 = 0;

      if (actual_engine_pct_trq < 5)
         predip_timer_1 = PREDIP_NORMAL_TIME;
      else
         desired_engine_pct_trq = actual_engine_pct_trq;
   }

   engine_control = TORQUE_CONTROL;
   command_ETC1 = C_ETC1_OVERSPEED;

   if (predip_timer_1 < PREDIP_NORMAL_TIME)
   {
      if ((desired_engine_pct_trq >= TORQUE_RAMP_OFF_RATE) &&
         (actual_engine_pct_trq > 0))
      {
         desired_engine_pct_trq -= TORQUE_RAMP_OFF_RATE;
      }
      else
      {
         desired_engine_pct_trq = 0;

         /* check to force bump if neutral not achieved */
         if (actual_engine_pct_trq < 10)
         {
            if (++predip_timer_3 >= PREDIP_ZERO_FDBK_TIME)
               predip_timer_1 = PREDIP_NORMAL_TIME;
         }
      }
      ++predip_timer_1;
   }
   else
   {
      if ((lpf_output_accel > -150) &&
         (predip_timer_1 < (PREDIP_NORMAL_TIME + PREDIP_TORQUE_ZERO_TIME)))
      {
         predip_torque_bump_time = PREDIP_TORQ_BUMP_TIME_LO;
         predip_torque_bump_value = PREDIP_TORQ_BUMP_VALUE_LO + needed_percent_for_zero_flywheel_trq;
      }
      else
      {
         if (predip_timer_1 < (PREDIP_NORMAL_TIME + 2*PREDIP_TORQUE_ZERO_TIME))
         {
            predip_torque_bump_time = PREDIP_TORQ_BUMP_TIME_MED;
            predip_torque_bump_value = PREDIP_TORQ_BUMP_VALUE_MED + needed_percent_for_zero_flywheel_trq;
         }
         else
         {
            predip_torque_bump_time = PREDIP_TORQ_BUMP_TIME_HI;
            predip_torque_bump_value = PREDIP_TORQ_BUMP_VALUE_HI + needed_percent_for_zero_flywheel_trq;
         }
      }

      if (predip_timer_2 < predip_torque_bump_time)
      {
         desired_engine_pct_trq = predip_torque_bump_value;
         if (actual_engine_pct_trq > 0)
         {
```

```
            ++predip_timer_1;
            ++predip_timer_2;
         }
      }
      else
      {
         desired_engine_pct_trq = 0;
         ++predip_timer_1;
         ++predip_timer_2;
      }

      if (predip_timer_2 >= PREDIP_TORQUE_ZERO_TIME)
         predip_timer_2 = 0;
   }
}
#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_sync_lever    (AutoSplit)
 *
 * Description:
 *   This function synchronizes engine speed to output shaft speed
 *   during a shift.
 *
 *****************************************************************************/

static void control_engine_sync_lever(void)
{

  if (accelerator_pedal_position > THREE_PERCENT)
    sync_maintain_timer = MAINTAIN_SYNC_TIME;

  if ((engine_status != ENGINE_SYNC_MODE) || (sync_maintain_timer == 0))
  {
    sync_on_timer = 0;
    sync_off_timer = 0;

    if (engine_status != ENGINE_SYNC_MODE)  /* first time through sync */
    {
      sync_maintain_timer = MAINTAIN_SYNC_TIME;
      engine_status = ENGINE_SYNC_MODE;
    }
    else                          /* sync_maintain_timer reached 0 */
    {
      engine_control = OVERRIDE_DISABLED;
      command_ETC1 = C_ETC1_NORMAL;
    }
  }

  else
  {
    sync_maintain_timer--;

    if (sync_on_timer++ <= 296)      /* allow sync mode for about 3 SEC */
    {
      sync_off_timer = 0;
      engine_control = SPEED_CONTROL;
      command_ETC1 = C_ETC1_OVERSPEED;

      if (shift_type == UPSHIFT)
      {
        sync_offset = -65;                                  /* RPM */
      }

      else   /* shift is a downshift */
      {
        sync_offset = -65;                                  /* RPM */
      }


      if (gos_signed + sync_offset > 0)
      {
      /*  desired_engine_speed = (int)(gos_signed + sync_offset); */

        _cx = dos_filtered;                                 /* BIN 0 */
        _bx = trn_tbl.gear_ratio[destination_gear + GR_OFS];  /* BIN 8 */
        _ax = sync_dos_offset_K1;                           /* BIN 8 */
        asm mul _cxdx, _bx;                                 /* BIN 8 */
        asm div _cxdx, _ax;               /* divide by constant BIN 8 */
        sync_dos_offset = _cx;            /* save final result  BIN 0 */

        desired_engine_speed = (int)(gos_signed + sync_offset + sync_dos_offset);


#if (0)
        desired_eng_spd_new = (int)(gos_signed + sync_offset);

        if (desired_eng_spd_new > desired_engine_speed)
          desired_eng_spd_diff = desired_eng_spd_new - desired_engine_speed;
        else
          desired_eng_spd_diff = desired_engine_speed - desired_eng_spd_new;

        if (desired_eng_spd_diff > desired_eng_spd_delta)
           desired_engine_speed = desired_eng_spd_new;
#endif
```

```
      )

      else
        desired_engine_speed = 0;
    )
    else
    (
      if (sync_off_timer <= 4)
      (
        sync_off_timer++;
#if (0)
        engine_control = TORQUE_CONTROL;
        command_ETC1 = C_ETC1_OVERSPEED;
        desired_engine_pct_trq = needed_percent_for_zero_flywheel_trq;
#endif
      )
      else
        sync_on_timer = 0;

    )
  )
)

#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_sync_auto    (AutoSplit)
 *
 * Description:
 *   This function synchronizes engine speed to output shaft speed
 *   during a shift.
 *
 *****************************************************************************/

static void control_engine_sync_auto(void)
{
  if (accelerator_pedal_position > THREE_PERCENT)
    sync_maintain_timer = MAINTAIN_SYNC_TIME;

  if ((engine_status != ENGINE_SYNC_MODE) || (sync_maintain_timer == 0))
  {
    sync_on_timer = 0;
    sync_off_timer = 0;
    sync_first_pass = TRUE;
    sync_first_pass_timer = SYNC_FIRST_PASS_TIME;

    if (shift_type == UPSHIFT)
      sync_offset = -65;
    else
      sync_offset = 65;

    if (engine_status != ENGINE_SYNC_MODE)  /* first time through sync */
    {
      sync_maintain_timer = MAINTAIN_SYNC_TIME;
      engine_status = ENGINE_SYNC_MODE;
    }
    else                          /* sync_maintain_timer reached 0 */
    {
      engine_control = OVERRIDE_DISABLED;
      command_ETC1 = C_ETC1_NORMAL;
    }
  }

  else
  {
    sync_maintain_timer--;

    if (sync_on_timer++ <= 200)      /* allow sync mode for about 2 seconds */
    {
      sync_off_timer = 0;
      engine_control = SPEED_CONTROL;
      command_ETC1 = C_ETC1_OVERSPEED;

      if (sync_first_pass == TRUE)
      {
        if (shift_type == UPSHIFT)
        {
          sync_speed_modified = (signed int)(input_speed) +
                                (input_speed_accel_filtered /(1000/ENG_RESPONSE_UPSHF_TIME));

          if (sync_speed_modified < gos_signed)
          {
            if (sync_first_pass_timer == 0)
            {
              sync_offset = 65;
              sync_first_pass = FALSE;
            }
            else
              sync_first_pass_timer--;
          }
        }
        else   /* shift is a downshift */
        {
          sync_speed_modified = (signed int)(input_speed) +
                                (input_speed_accel_filtered /(1000/ENG_RESPONSE_DNSHF_TIME));

          if (sync_speed_modified > gos_signed)
          {
          /* if (sync_first_pass_timer == 0)  */
          /*  {                               */
              sync_first_pass = FALSE;
              if (pct_demand_at_cur_sp < 15)
                sync_offset = -65;
```

```
          /*  }                                   */
          /*  else                                */
          /*    sync_first_pass_timer--;          */
          }
        }
      }

      if (gos_signed + sync_offset > 0)
        desired_engine_speed = (int)(gos_signed + sync_offset);
      else
        desired_engine_speed = 0;
    }
    else
    {
      if (sync_off_timer <= 4)
      {
        sync_off_timer++;
#if (0)
        engine_control = TORQUE_CONTROL;
        command_ETC1 = C_ETC1_OVERSPEED;
        desired_engine_pct_trq = needed_percent_for_zero_flywheel_trq;
#endif
      }
      else
      {
        sync_on_timer = 0;
        sync_offset = -(sync_offset);   /* force sync speed to toggle around gos */
      }
    }
  }
}
#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_sync    (AutoSplit)
 *
 * Description:
 *   This function synchronizes engine speed to output shaft speed
 *   during a shift.
 *
 *****************************************************************************/

static void control_engine_sync(void)
{
   if (shift_init_type == AUTO)
     control_engine_sync_auto();
   else
     control_engine_sync_lever();

   intent_to_shift = FALSE;

}

#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_sync_test_mode    (AutoSplit)
 *
 * Description:
 *   This function test the synchronize mode of engine speed control.
 *
 *****************************************************************************/

static void control_engine_sync_test_mode(void)
{

      if (accelerator_pedal_position < 10)
      {
        engine_status = ENGINE_FOLLOWER_MODE;
        engine_commands = ENGINE_FOLLOWER;
        engine_control = OVERRIDE_DISABLED;
        command_ETC1 = C_ETC1_NORMAL;
        desired_engine_speed = 0;
      }
      else
      {
        if (accelerator_pedal_position > 90)
        {
          engine_status = ENGINE_SYNC_MODE;
          engine_commands = ENGINE_SYNC;
          engine_control = SPEED_CONTROL;
          command_ETC1 = C_ETC1_OVERSPEED;
          desired_engine_speed = desired_engine_speed_test;
          desired_engine_speed_timer = desired_engine_speed_time;
        }
        else
        {
          if (desired_engine_speed_timer > 0)
             desired_engine_speed_timer--;
          else
          {
            if (desired_engine_speed > 600)
            {
              desired_engine_speed_timer = desired_engine_speed_time;
              desired_engine_speed = (desired_engine_speed - desired_engine_speed_ramp);
            }
          }
        }
      }
}
#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  determine_if_recovery_complete
 *
 *   Description:
 *      This routine checks to see if the percent_torque_value_limit has
 *      exceeded the percent_torque_value feedback from the engine by x%
 *      for x milliseconds and will then set percent_torque_value_limit
 *      to 100% to cancel the recovery mode.
 *
 *****************************************************************************/

static void determine_if_recovery_complete(void)
{
   if ((net_engine_pct_trq > 10) &&
      (desired_engine_pct_trq > (net_engine_pct_trq + RECOVERY_CANCEL_OFFSET)))
   {
      ++recovery_cancel_timer;
   }
   else
      recovery_cancel_timer = 0;

   if ( (recovery_cancel_timer >= RECOVERY_CANCEL_TIME) ||
      (desired_engine_pct_trq == 100) )
   {
      /* terminate the recovery mode */
      desired_engine_pct_trq = 100;
      engine_status = ENGINE_RECOVERY_MODE_COMPLETE;
   }
}


#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_recovery_normal
 *
 * Description:
 *      Determine throttle command for recovery mode.
 *      TORQUE_LIMIT is scaled as a BIN 8 number representing the percentage
 *      of torque allowed to the engine during recovery.
 *
 *****************************************************************************/

static void control_engine_recovery_normal(void)
{
   engine_control = SPEED_TORQUE_LIMIT;
   command_ETC1 = C_ETC1_NORMAL;

   desired_engine_speed = 8031;   /* torque limit only, max value for speed */

   torque_limit += RECOVERY_RATE_TABLE[destination_gear+4];   /* BIN 8 */

   desired_engine_pct_trq = (char)(torque_limit >> 8);   /* BIN 0 */

   determine_if_recovery_complete();
}


#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_recovery_coasting
 *
 * Description:
 *      Determine throttle command for coasting down shifts mode.
 *
 *****************************************************************************/

static void control_engine_recovery_coasting(void)
{
   register uint local_uint;

   if (sync_on_timer <= 300)
   {
      ++sync_on_timer;

      engine_control = SPEED_CONTROL;
      command_ETC1 = C_ETC1_NORMAL;

      sync_off_timer = 0;

      /** recov_coast_down_tmp1 = gos + (dgos * K1) - THLO_DS_ENG_DECAY_RAMP **/

      if (dgos < 0)         /* get absolute value */
         _cx = (uint)-dgos;
      else
         _cx = (uint)dgos;

      asm mulu _cxdx, #THLO_DS_ENG_DECAY_K1;      /* BIN 12 */
      asm shrl _cxdx, #12;                        /* BIN 0 */

      if (_cxdx > 500)                           /* error check */
         local_uint = 0;
      else
         local_uint = _cx;

      if (lpf_output_accel > 0)
         recov_coast_down_tmp1 = (gos + local_uint) - THLO_DS_ENG_DECAY_RAMP;
      else
         recov_coast_down_tmp1 = (gos - local_uint) - THLO_DS_ENG_DECAY_RAMP;

      /** recov_coast_down_tmp2 = desired_engine_speed - THLO_DS_ENG_DECAY_RAMP **/

      recov_coast_down_tmp2 = desired_engine_speed - THLO_DS_ENG_DECAY_RAMP;

      if (recov_coast_down_tmp1 < recov_coast_down_tmp2)
         desired_engine_speed = recov_coast_down_tmp1;
      else
         desired_engine_speed = recov_coast_down_tmp2;
   }
   else
   {
      if (sync_off_timer <= 5)
      {
         ++sync_off_timer;
         engine_control = TORQUE_CONTROL;
         command_ETC1 = C_ETC1_NORMAL;
         desired_engine_pct_trq = 0;
      }
      else
         sync_on_timer = 0;
   }

   if ((desired_engine_speed + THLO_DS_FINISHED_DELTA) < gos)
   {
      /* terminate the recovery mode */
      desired_engine_pct_trq = 100;
      engine_status = ENGINE_RECOVERY_MODE_COMPLETE;
   }
}


#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_recovery
 *
 * Description:
 *      This function determines which type of throttle recovery should be
 *      used.  And initializes some of the variables that will be used.
 *
 *****************************************************************************/

static void control_engine_recovery(void)
{

   if ((engine_status != ENGINE_RECOVERY_MODE) &&
      (engine_status != ENGINE_RECOVERY_MODE_COMPLETE))
   {
      engine_status = ENGINE_RECOVERY_MODE;
      desired_engine_pct_trq = 0;
      recovery_cancel_timer = 0;
      sync_on_timer = 0;
      sync_off_timer = 0;

      /* kill pedal transition stuff */
      positive_pedal_trans = FALSE;
      positive_pedal_trans = FALSE;
      zero_flywheel_trq_timer = 0;
      zero_flywheel_trq_time = 0;


      if (gos < desired_engine_speed)
         desired_engine_speed = gos;

      /* set initial starting torque limit */      /* percent, BIN 8 */
      if ((actual_engine_pct_trq > needed_percent_for_zero_flywheel_trq) &&
          (pct_demand_at_cur_sp > 5))
        torque_limit = ((unsigned int)(actual_engine_pct_trq))<<8; /* percent, BIN 8 */
      else
        torque_limit = ((unsigned int)(needed_percent_for_zero_flywheel_trq))<<8; /* percent, BIN 8 */
   }

   if ((destination_gear > ACTIVE_RECOVERY_GEAR) &&
      (pct_demand_at_cur_sp < 5) &&
      ((shift_type == COAST_DOWN_SHIFT) ||
      (shift_type == UPSHIFT)))
   {
      control_engine_recovery_coasting();
   }
   else
   {
      control_engine_recovery_normal();
   }
}
#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_intent_to_shift
 *
 * Description:
 *      This function
 *
 *****************************************************************************/

static void control_intent_to_shift(void)
{

   if (engine_control != TORQUE_CONTROL)
   {
      desired_engine_pct_trq = actual_engine_pct_trq;
   }

   engine_control = TORQUE_CONTROL;
   command_ETC1 = C_ETC1_OVERSPEED;

   positive_pedal_trans = TRUE;  /* allow for recovery mode */

   if ((desired_engine_pct_trq >= intent_ramp_off_rate) &&
      (actual_engine_pct_trq > intent_final_pct_trq))
   {
      desired_engine_pct_trq -= intent_ramp_off_rate;
   }
   else
   {
      desired_engine_pct_trq = intent_final_pct_trq;
   }

}

#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_follower
 *
 * Description:
 *      This function sets the override_control_mode to no over ride so that
 *      the engine follows the accelerator demand.
 *
 *****************************************************************************/

static void control_engine_follower(void)
{

#define POSITIVE_PEDAL_TRANSITION_TIME     25  /* 250 MSEC */
#define NEGATIVE_PEDAL_TRANSITION_TIME     40  /* 400 MSEC */


  engine_status = ENGINE_FOLLOWER_MODE;

  if ((intent_to_shift == TRUE) &&
      (shift_in_process == FALSE) &&
      (desired_gear != destination_gear_selected))
  {
    control_intent_to_shift();
  }

  else
  {
    if ((accelerator_pedal_position >= 5) &&     /* positive pedal transition */
        (accelerator_pedal_position_old <= 4) &&
        (low_speed_latch == FALSE))
    {
      positive_pedal_trans = TRUE;
      zero_flywheel_trq_time = POSITIVE_PEDAL_TRANSITION_TIME;
      if (zero_flywheel_trq_timer >= NEGATIVE_PEDAL_TRANSITION_TIME)
        zero_flywheel_trq_timer = 0;
    }

    else
    {
      if ((accelerator_pedal_position <= 4) &&   /* negative pedal transition */
          (accelerator_pedal_position_old >= 5) &&
          (low_speed_latch == FALSE))
      {
        zero_flywheel_trq_time = NEGATIVE_PEDAL_TRANSITION_TIME;
        zero_flywheel_trq_timer = 0;
      }
    }

    if ((zero_flywheel_trq_timer < zero_flywheel_trq_time) &&
        (current_gear > 1) && (current_gear < 10) && (low_speed_latch == FALSE))
    {
      engine_control = TORQUE_CONTROL;
      command_ETC1 = C_ETC1_OVERSPEED;
      desired_engine_pct_trq = needed_percent_for_zero_flywheel_trq;

      if (actual_engine_pct_trq < (needed_percent_for_zero_flywheel_trq + 5));
        zero_flywheel_trq_timer++;
    }

    else
    {
      if ((positive_pedal_trans == TRUE) && (low_speed_latch == FALSE))
      {
        positive_pedal_trans = FALSE;
        engine_commands = ENGINE_RECOVERY;       /* engine: finish torque return */
        control_engine_recovery();
      }

      else
      {
        engine_control = OVERRIDE_DISABLED;
        command_ETC1 = C_ETC1_NORMAL;
      }
    }
  }  /* end no intent_to_shift */
}
#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_idle
 *
 * Description:
 *      This function sets the engine controls for the idle mode.
 *
 *****************************************************************************/

static void control_engine_idle(void)
{
   engine_control = OVERRIDE_DISABLED;
   command_ETC1 = C_ETC1_NORMAL;

   engine_status = ENGINE_IDLE_MODE;
}


#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_start
 *
 * Description:
 *      This function sets the engine controls for the start mode.
 *
 *****************************************************************************/

static void control_engine_start(void)
{
   engine_control = OVERRIDE_DISABLED;
   command_ETC1 = C_ETC1_NORMAL;

   engine_status = ENGINE_START_MODE;
}


#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  control_engine_compression_brake
 *
 * Description:
 *      This function controls the state of the engine compression brake.
 *      The brake can be used during upshifts to speed up the decel rate of
 *      the input shaft.
 *
 *****************************************************************************/

static void control_engine_compression_brake(void)
{
   if (engine_communication_active &&
      (engine_status == ENGINE_SYNC_MODE) &&
      (shift_type == UPSHIFT) &&
      (input_speed_filtered > (gos + 150)) &&
      (destination_gear > 1) &&
      (destination_gear < 7) &&
      (engine_brake_available) &&
      ((dos_predicted < dos_prdtd_lim_no_jake) || eng_brake_assist))
   {
      eng_brake_assist = TRUE;
   }
   else
   {
      eng_brake_assist = FALSE;
   }


 eng_brake_assist = FALSE;   /* force false state for now */
}


#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  determine_gos
 *
 * Description:
 *      This function mulitplies the destination gear ratio times the
 *      output shaft speed for use in the DRL_CMDS module.
 *
 *                  gos = (g)ear * (o)utput (s)peed
 *
 *****************************************************************************/

static void determine_gos(void)
{

  /*** determine gos for the destination_gear ***/
   _bx = trn_tbl.gear_ratio[destination_gear + GR_OFS];
   _cx = output_speed_filtered;   /* output speed */
   asm mulu _cxdx, _bx;            /* BIN 8 result */
   asm shrl _cxdx, #8;            /* make BIN 0 */
   gos = _cx;                     /* BIN 0 */

   _cx = *(uint *)&lpf_output_accel;
   asm mul _cxdx, _bx;
   asm div _cxdx, #256;
   dgos = *(int *)&_cx;

  gos_signed = (signed int)(gos);   /* allow signed math in other functions*/

  /*** determine gos for the "current_gear" ***/
   _bx = trn_tbl.gear_ratio[current_gear + GR_OFS];
   _cx = output_speed_filtered;   /* output speed */
   asm mulu _cxdx, _bx;            /* BIN 8 result */
   asm shrl _cxdx, #8;            /* make BIN 0 */
   gos_current_gear = _cx;         /* BIN 0 */

}

#pragma EJECT
```

```
/*****************************************************************************
 *
 * Function:  determine_shiftability_variables
 *
 * Description:
 *      This function filters both the output speed and the rate of change of
 *      the output speed for use in the shiftability function. This function
 *      also calulates the rate of change of the input shaft based on the
 *      filtered value for the rate of change of the output shaft.
 *
 *      The filters used in this function are required to get a high level of
 *      stability.  The BIN 8 filter used here will provide a much smoother
 *      output which is needed to filter out driveline oscillations.
 *
 *      Note:  These calculations were placed in this module because it is
 *      called on a regular 10 msec interval.  These calculations should
 *      be placed in the pr_i_s_i.c96 module once shiftability is proven.
 *      These variables are used in the SEL_GEAR.C96 module.
 *
 *****************************************************************************/

static void determine_shiftability_variables(void)
{
   /* LPF coefficients: exp(-wT), T=0.010s */
   #define OS_LPF        248      /* 0.9691 BIN 8 (0.50Hz) */
   #define DOSFK1        249      /* 0.9727 BIN 8 (0.44Hz) */
   #define EPTFK1        252      /* 0.9844 BIN 8 (0.25Hz) */

   #define IS_FK1        236      /* 0.9219 BIN 8 (0.??Hz) */
   #define OS_FK1        236      /* 0.9219 BIN 8 (0.??Hz) */
   #define DISFK1        236      /* 0.9219 BIN 8 (0.??Hz) */

  #define LOW_RANGE     3197      /* 3.1224 BIN 10         */
  #define BIN_10        1024

   static long dos_filtered_bin8;
   static int ept_filtered_bin8;

unsigned long is_filtered_partial_1;
unsigned long is_filtered_partial_2;
unsigned long os_filtered_partial_1;
unsigned long os_filtered_partial_2;

   /** create lpf_output_accel **/

   _bx = *(uint *)&output_speed_accel;            /* _bx = x(n), BIN 0 */
   _cx = *(uint *)&lpf_output_accel - _bx;        /* _cx = y(n-1) - x(n), BIN 0 */
   asm mul _cxdx, #OS_LPF;                         /* _cxdx = K*(...), BIN 8 */
   asm div _cxdx, #256;                            /* make BIN 0 */
   _bx += _cx;                                     /* _bx = x(n) + K*(...), BIN 0 */
   lpf_output_accel = *(int *)&_bx;                /* save acceleration */

   /** dos_filtered = (dos_filtered * DOSFK1) + (lpf_output_accel * (1-DOSFK1) **/

   _cxdx = *(ulong *)&dos_filtered_bin8;           /* BIN 8 */
   asm shral _cxdx, #2;                            /* BIN 6 (_cx) */
   asm mul _cxdx, #DOSFK1;                         /* BIN 14 */
   asm shral _cxdx, #6;                            /* BIN 8 */
   dos_filtered_bin8 = *(long *)&_cxdx;            /* save partial result */

   _cx = *(uint *)&lpf_output_accel;               /* BIN 0 */
   _bx = 256 - DOSFK1;                             /* 1 BIN 8 - DOSFK1 */
   asm mul _cxdx, _bx;                             /* BIN 8 */
   dos_filtered_bin8 += *(long *)&_cxdx;           /* sum is final result */

   dos_filtered = (int)(dos_filtered_bin8 >> 8);   /* BIN 0 */

   /** eng_percent_torque_filtered = (eng_percent_torque_filtered * EPTFK1) +
                   (net_engine_pct_trq * (1-EPTFK1) **/

   _cx = *(uint *)&ept_filtered_bin8;              /* BIN 8 */
   asm mul _cxdx, #EPTFK1;                         /* BIN 16 */
   asm shral _cxdx, #8;                            /* BIN 8 */
   ept_filtered_bin8 = *(int *)&_cx;               /* save partial result */

   _cx = net_engine_pct_trq;                       /* BIN 0 */
   _bx = 256 - EPTFK1;                             /* 1 BIN 8 - EPTFK1 */
   asm mul _cxdx, _bx;                             /* BIN 8 */
   ept_filtered_bin8 += *(int *)&_cx;              /* sum is final result */
```

```
eng_percent_torque_filtered = (char)(ept_filtered_bin8 >> 8);

/** input_shaft_accel_calculated = dos_filtered * gear_ratio **/

_cx = trn_tbl.gear_ratio[destination_gear + GR_OFS];   /* BIN 8 */
_bx = *(uint *)&dos_filtered;                          /* BIN 0 */
asm mul _cxdx, _bx;                                    /* BIN 8 */
asm shral _cxdx, #8;                                   /* BIN 0 */
input_shaft_accel_calculated = *(int *)&_cx;

/***  calculate filtered input and output shaft speeds for AutoSplit  ***/

/***  determine os_based_on_rcs variable  ***/
if (output_speed < 1000)
{
  _bx = aux_speed;                                     /* BIN 0 */
  _cx = BIN_10;                                        /* BIN 10 */
  _ax = LOW_RANGE;                                     /* BIN 10 */
  asm mulu _cxdx, _bx;                                 /* make aux_speed BIN 10 */
  asm divu _cxdx, _ax;                                 /* divide by low range BIN 10 */
  os_based_on_rcs = _cx;                               /* BIN 0 */
}

/** input_speed_filtered = (input_speed_filtered * IS_FK1) +
                  (input_speed * (1-IS_FK1) **/

_ax = (is_filtered_bin8 >> 4) ;                        /* BIN 4          */
_cx = IS_FK1 ;                                         /* BIN 8          */
asm mulu _axbx, _cx ;                                  /* BIN 12         */
asm shrl _axbx, #4 ;                                   /* BIN 8          */
is_filtered_partial_1 = _axbx ;                        /* BIN 8          */

_cx = input_speed ;                                    /* BIN 0          */
_ax = 256 - IS_FK1 ;                                   /* 1 BIN 8 - IS_FK1 */
asm mulu _axbx, _cx ;                                  /* BIN 8          */
is_filtered_partial_2 = _axbx ;                        /* BIN 8          */

is_filtered_bin8 =  is_filtered_partial_1 + is_filtered_partial_2 ;

input_speed_filtered = (unsigned int)(is_filtered_bin8 >> 8);  /* BIN 0          */


/** output_speed_filtered = (output_speed_filtered * OS_FK1) +
                  (output_speed * (1-OS_FK1) **/

_ax = (os_filtered_bin8 >> 4) ;                        /* BIN 4          */
_cx = OS_FK1 ;                                         /* BIN 8          */
asm mulu _axbx, _cx ;                                  /* BIN 12         */
asm shrl _axbx, #4 ;                                   /* BIN 8          */
os_filtered_partial_1 = _axbx ;                        /* BIN 8          */

if (output_speed < 250)
  _cx = os_based_on_rcs;                               /* BIN 0          */
else
  _cx = output_speed ;                                 /* BIN 0          */

_ax = 256 - OS_FK1 ;                                   /* 1 BIN 8 - OS_FK1 */
asm mulu _axbx, _cx ;                                  /* BIN 8          */
os_filtered_partial_2 = _axbx ;                        /* BIN 8          */

os_filtered_bin8 =  os_filtered_partial_1 + os_filtered_partial_2 ;

output_speed_filtered = (unsigned int)(os_filtered_bin8 >> 8);  /* BIN 0          */


/** input_speed_accel_filtered = (input_speed_accel_filtered * DISFK1) + (input_shaft_accel * (1-DISFK1) **/

_cxdx = *(ulong *)&dis_filtered_bin8;                  /* BIN 8              */
asm shral _cxdx, #4;                                   /* BIN 4 (_cx)        */
asm mul _cxdx, #DISFK1;                                /* BIN 12             */
asm shral _cxdx, #4;                                   /* BIN 8              */
dis_filtered_bin8 = *(long *)&_cxdx;                   /* save partial result */

_cx = *(uint *)&input_speed_accel;                     /* BIN 0              */
_bx = 256 - DISFK1;                                    /* 1 BIN 8 - DISFK1   */
asm mul _cxdx, _bx;                                    /* BIN 8              */
dis_filtered_bin8 += *(long *)&_cxdx;                  /* sum is final result */

input_speed_accel_filtered = (int)(dis_filtered_bin8 >> 8);   /* BIN 0 */
```

```
   /** determine state of clutch **/

   if (engine_speed > input_speed)
     clutch_slip_speed = engine_speed - input_speed;
   else
     clutch_slip_speed = input_speed - engine_speed;

  if (clutch_slip_speed > 200)
     clutch_state = DISENGAGED;
  else
     if ((engine_speed > 800) && (low_speed_latch == FALSE))
       clutch_state = ENGAGED;


  /** determine desired percent torque needed for zero torque at flywheel **/

#if (0)
  if ((accelerator_pedal_position < 2) &&
      (clutch_state == ENGAGED) &&
      (current_gear == 0) &&
      (input_speed_filtered < 1100) &&
      (((engine_control == OVERRIDE_DISABLED) &&
        (low_speed_latch == FALSE) && (current_gear == 0)) ||
        (output_speed_filtered < 20)))
    percent_torque_accessories = eng_percent_torque_filtered;  /* get at idle */
#endif
    percent_torque_accessories = 3;  /* force value for now */

  needed_percent_for_zero_flywheel_trq = percent_torque_accessories +
                                         nominal_friction_pct_trq;

   overall_error = ((signed int)(input_speed_filtered) - (signed int)(gos));

}


#pragma EJECT
```

calc. for zero torque flywheel

flywheel torque = gross engine torque − (base friction torque + engine driven torque + engine friction torque)

Equivalent to col. 4, ln 3 of .477

```
/*****************************************************************************
 *
 * Function:  communicate_with_driveline
 *
 * Description:
 *      This is the periodic task which controls the actions of the engine
 *      by defining mode of control and controlling speed and torque output
 *      levels depending upon the control function being performed. This task
 *      is also intended for control of other driveline components (not yet
 *      named) which may be available in the future.
 *
 *****************************************************************************/

void communicate_with_driveline(void)
{
   initialize_driveline_data();

   x_start_periodic();
   while (1)
   {
      control_engine_compression_brake();

      determine_gos();   /* calculate (G)ear times the (O)utput (S)haft */

      determine_shiftability_variables();

      if (engine_communication_active)
      {

        if ((desired_sync_test_mode == TRUE) && (output_speed_filtered < 100))
          control_engine_sync_test_mode();

        else
        {      /* start of normal engine_commands switch */

         switch (engine_commands)
         {
         case ENGINE_PREDIP:
            control_engine_predip();
            break;

         case ENGINE_SYNC:
            control_engine_sync();
            break;

         case ENGINE_RECOVERY:
            control_engine_recovery();
            break;

         case ENGINE_IDLE:
            control_engine_idle();
            break;

         case ENGINE_START:
            control_engine_start();
            break;

         case ENGINE_FOLLOWER:
         default:
            control_engine_follower();
            break;
         }
        }      /* end of normal engine_commands switch */

         switch (eng_brake_command)
         {
         case ENG_BRAKE_OFF:
            retarder_control = TORQUE_CONTROL;
            desired_retarder_pct_trq = 0;
            break;

         case ENG_BRAKE_FULL:
            retarder_control = TORQUE_CONTROL;
            desired_retarder_pct_trq = -100;
            break;

         case ENG_BRAKE_IDLE:
         default:
            retarder_control = OVERRIDE_DISABLED;
```

```
            desired_retarder_pct_trq = 0;
            break;
         }
      }
      else
         engine_status = ENGINE_NOT_PRESENT;

      /* store old value for use in "control_engine_follower" function */
      accelerator_pedal_position_old = accelerator_pedal_position;

      x_sync_periodic(US_PER_LOOP);
   }
   x_end_periodic();
}
```

```
/*****************************************************************************
 *
 *        Unpublished and confidential.  Not to be reproduced,
 *        disseminated, transferred or used without the prior
 *        written consent of Eaton Corporation.
 *
 *        Copyright Eaton Corporation, 1994
 *        All rights reserved.
 *
 *****************************************************************************
 *
 * Filename:  pr_s_i_s.c96    (AutoSplit)
 *
 * Description:
 *   The modules contained within this compilation unit are
 *   intended to implement functionality of the Process System
 *   Input Signals task defined in the design documention.
 *   In general, Analog to digital conversions are started on
 *   Port0. The necessary hardware initialization and variable
 *   initializtion for inputs on Port0 are handled.  The switch
 *   inputs are captured to avoid conflict with AD conversions
 *   and all necessary scaling and error check for these inputs
 *   is conducted.
 *
 * Part Number:  <none>
 *
 * $Log:   ?
 *
 *    Rev 1.1   19 May 1994 11:32:26   markyvech
 * Converted for use with AutoSplit ECU2
 *
 *    Rev 1.0   12 Sep 1991 08:04:26   amsallen
 * Initial revision.
 *
 *****************************************************************************/


/*****************************************************************************
 *
 * Header files included.
 *
 *****************************************************************************/

#include <exec.h>            /* executive information                    */
#include <kr_sfr.h>          /* KR special function registers            */
#include <kr_def.h>          /* KR definitions                           */
#include <c_regs.h>          /* KR internal register definitions         */
#include <wwslib.h>          /* world wide software definitions          */
#include "pr_s_i_s.h"        /* process system input signal information */
#include "sysgen.h"          /* defines the task names and priority      */


/*****************************************************************************
 *
 * #defines local to this file.
 *
 *****************************************************************************/

/* Start_AD_Conversions */

#define ENABLE_AD_PTS_SCAN 0X20
#define ENABLE_AD_ISR 0X20

#define PERIOD 10U                                   /* 10ms */
#define RKM_PERIOD 50U                               /* 50ms */


/*****************************************************************************
 *
 * Constants and variables declared by this file.
 *
 *****************************************************************************/

/* Analog Inputs on Port0 */

int ignition_volts;
int splitter_position;

#define IGNITION_VOLTS_CHANNEL_RESULT   1
```

```
#define SPLITTER_POS_CHANNEL_RESULT    15

#define CONVERSION_TIME 0xef           /* for state time = 125 nsec:     */
                                       /*    sample time = 3.6250 usec   */
                                       /*    convert time = 20.1875 usec */
#define CONVERT_8 8                    /* scan and convert 8 channels    */

#define CONVERT_IGNITION_VOLTS         (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_0)
#define UNUSED_CHANNEL_1               (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_1)
#define UNUSED_CHANNEL_2               (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_2)
#define UNUSED_CHANNEL_3               (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_3)
#define UNUSED_CHANNEL_4               (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_4)
#define UNUSED_CHANNEL_5               (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_5)
#define UNUSED_CHANNEL_6               (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_6)
#define CONVERT_SPLITTER_POS           (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_7)
#define STOP_CONVERSION                0x00
#define START_CONVERSIONS              _ad_command = CONVERT_IGNITION_VOLTS

/* table containing AD_result and AD_Command values after PTS scan */
unsigned int AD_Table[16];

/* AD SCAN PTS CONTROL BLOCK LOCATION */
_ad_ptsb_type AD_Con_Block;
#pragma locate(AD_Con_Block=0x01F8)    /* locate pts control block */
#pragma pts(AD_Con_Block = 5)          /* set pts vector 5, A/D done */


#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  Initialize_Input_Signals
 *
 * Description:
 *    This routine initializes the A/D converter.  It sets the A/D to
 *    run in PTS scan mode, 10bit conversion. The PTS control block is
 *    set up and the Command/result table is initialized.
 *
 *****************************************************************************/

void Initialize_Input_Signals(void)
{
   /* if we knew when the first speed packet arrived, we could initialize
      with those values.  since we don't, be safe and use zero. */

   AD_Table[0]  = UNUSED_CHANNEL_1;          /* place holder for channel 1     */
   AD_Table[1]  = 0x0000;                    /* IGNITION_VOLTS_CHANNEL_RESULT  */
   AD_Table[2]  = UNUSED_CHANNEL_2;          /* place holder for channel 2     */
   AD_Table[3]  = 0x0000;                    /* UNUSED_1_RESULT                */
   AD_Table[4]  = UNUSED_CHANNEL_3;          /* place holder for channel 3     */
   AD_Table[5]  = 0x0000;                    /* UNUSED_2_RESULT                */
   AD_Table[6]  = UNUSED_CHANNEL_4;          /* place holder for channel 4     */
   AD_Table[7]  = 0x0000;                    /* UNUSED_3_RESULT                */
   AD_Table[8]  = UNUSED_CHANNEL_5;          /* place holder for channel 5     */
   AD_Table[9]  = 0x0000;                    /* UNUSED_4_RESULT                */
   AD_Table[10] = UNUSED_CHANNEL_6;          /* place holder for channel 6     */
   AD_Table[11] = 0x0000;                    /* UNUSED_5_RESULT                */
   AD_Table[12] = CONVERT_SPLITTER_POS;      /* Command convert splitter pos   */
   AD_Table[13] = 0x0000;                    /* UNUSED_6_RESULT                */
   AD_Table[14] = STOP_CONVERSION;           /*  command to Stop conversions   */
   AD_Table[15] = 0x0000;                    /* SPLITTER_POS_CHANNEL_RESULT    */

   AD_Con_Block.cnt = CONVERT_8;
   AD_Con_Block.ctrl = _AD_MODE|_S_D_UPDT;   /* A/D mode bits 0,1 of PTS_CONTROL */
                                             /* always set to 3h bit 2 = 0       */
                                             /* S/D update at end of cycle       */
                                             /* bit 5 always 0                   */
                                             /* Set mode for AD SCAN             */

   AD_Con_Block.s_d = AD_Table;              /* Load s_d with AD_Table address   */
   AD_Con_Block.reg = (void *)&_ad_result;   /* Load reg with AD_Result address  */

   _ad_time = CONVERSION_TIME;
   _ad_test = _NO_OFFS;                      /* Disable test mode */
   _pts_select &= ~(_PTS_ADDONE_BIT);        /* Disable AD PTS */

}


#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  AD_ISR
 *
 * Description:
 *    This interupt service routine resets the PTSCOUNT, pts_sd and pts_reg
 *    for another PTS_Scan A/D cycle. It also readies a task to run when
 *    a PTS cycle has completed.
 *
 *****************************************************************************/

#pragma interrupt(AD_ISR=5)
void AD_ISR(void)
{
   x_start_isr();
   AD_Con_Block.cnt = CONVERT_8;          /* Reset pts count for next cycle */
   AD_Con_Block.s_d = AD_Table;           /* Reset table pointer to start of table */
   AD_Con_Block.reg = (void *)&_ad_result;

   x_ready(PROCESS_INPUT_SIGNALS);        /* Ready pr_s_i_s task */
   x_end_isr();
}


#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  Start_AD_Conversions
 *
 * Description:
 *    This function initializes the input signal processing function
 *    if it has not already been done, and then startes the PTS Scan
 *    of the AD channels by sending the appropriate command to the
 *    ad_command register.
 *
 *****************************************************************************/

void Start_AD_Conversions(void)
{
   if ((_int_mask & ENABLE_AD_ISR) == 0 )
      Initialize_Input_Signals();            /* Set up AD table for PTS */

   _pts_select |= ENABLE_AD_PTS_SCAN;
   _int_mask |= ENABLE_AD_ISR;

   x_prearm_stimulus();

   START_CONVERSIONS;   /* Start a conversion, initiate the PTS cycles */

   x_wait_stimulus();   /* AD_ISR will ready task when PTS is complete */
}


#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  process_input_signals
 *
 * Description:
 *   This is the periodic task which starts the analog conversions on Port0.
 *
 *****************************************************************************/

void process_input_signals(void)
{
   x_start_periodic();
   while (1)
   {
      Start_AD_Conversions();

      /* Clean up and scale AD values */
      ignition_volts = scale_system_ad_inputs(IGNITION_VOLTAGE);
      splitter_position = scale_system_ad_inputs(SPLITTER_POSITION);

 /*     x_sync_periodic(PERIOD*1000); */
      x_sync_periodic(RKM_PERIOD*1000);
   }
   x_end_periodic();
}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  scale_system_ad_inputs
 *
 * Description:
 *    This function removes the channel, status and reserved bits from
 *    the raw AD values, and performs all necessary scaling and error
 *    checking for the analog inputs on Port0.
 *
 *****************************************************************************/

int scale_system_ad_inputs(char Channel)
{
   int Scaled_Value = 0;
   uint volts_per_bit;     /* BIN 16 */
   uint units_per_bit;     /* BIN 16 */
   #define TWELVE_VOLT_FULL_SCALE          22.46   /* volts */
   #define TWENTY_FOUR_VOLT_FULL_SCALE     40.49   /* volts */
   #define DISTANCE_FULL_SCALE            100      /*       */


   volts_per_bit = (uint)((TWELVE_VOLT_FULL_SCALE*65536/1023)+0.5);
   units_per_bit = (uint)((DISTANCE_FULL_SCALE*65536/1023)+0.5);

   switch (Channel)
   {
   case 0: /* IGNITION VOLTAGE */
      _cx = AD_Table[IGNITION_VOLTS_CHANNEL_RESULT] >> 6;
      asm mulu _cxdx, volts_per_bit;   /* volts, BIN 16 (_dx, BIN 0) */
      Scaled_Value = *(int *)&_dx;
      break;

   case 1: /* UNUSED */  /* to be completed when a product requires it */
      Scaled_Value = (0);
      break;

   case 2: /* UNUSED */  /* to be completed when a product requires it */
      Scaled_Value = (0);
      break;

   case 3: /* UNUSED */  /* to be completed when a product requires it */
      Scaled_Value = (0);
      break;

   case 4: /* UNUSED */  /* to be completed when a product requires it */
      Scaled_Value = (0);
      break;

   case 5: /* UNUSED */  /* to be completed when a product requires it */
      Scaled_Value = (0);
      break;

   case 6: /* UNUSED */  /* to be completed when a product requires it */
      Scaled_Value = (0);
      break;

   case 7: /* SPLITTER POSITION */
      _cx = AD_Table[SPLITTER_POS_CHANNEL_RESULT] >> 6;
      asm mulu _cxdx, units_per_bit;   /* distance, BIN 16 (_dx, BIN 0) */
      Scaled_Value = *(int *)&_dx;
      break;

   default:
      break;
   }
   return (Scaled_Value);
}
```

```
/*****************************************************************************
 *
 *        Unpublished and confidential.  Not to be reproduced,
 *        disseminated, transferred or used without the prior
 *        written consent of Eaton Corporation.
 *
 *        Copyright Eaton Corporation, 1993-94.
 *        All rights reserved.
 *
 *****************************************************************************
 *
 * Filename:  sel_gear.c96     (AutoSplit)
 *
 * Description:
 *   This module is the periodic task "select_gear".  It assigns values
 *   to destination_gear_selected as a function of selected_mode, input
 *   and output shaft speeds, and driver selections (dcc_manual_input).
 *   Shift parameters are in the data structure shf_tbl.  Before setting
 *   destination_gear_selected, gears are checked with available_gear().
 *
 * Part Number:  <none>
 *
 * $Log:   R:\ashift\vcs\sel_gear.c9v  $
 *
 *    Rev 1.8   21 Feb 1994 15:07:14   schroeder
 * Replaced shiftability_mode with new flag, engine_brake_available.
 *
 *    Rev 1.0   29 Jul 1993 16:40:26   schroeder
 * Initial revision.
 *
 *****************************************************************************/

/*****************************************************************************
 *
 * Header files included.
 *
 *****************************************************************************/

#include <exec.h>          /* executive information */
#include <c_regs.h>        /* c registers */
#include <wwslib.h>        /* contains common global defines */
#include "cont_sys.h"      /* control system */
#include "conj1939.h"      /* defines interface to j1939 control module */
#include "drl_cmds.h"      /* driveline commands information */
#include "sel_gear.h"      /* select gear */
#include "shf_tbl.h"       /* shift table definition */
#include "trn_tbl.h"       /* (system) transmission table definition */
#include "calc_spd.h"
#include "trns_act.h"
#pragma noreentrant

/*****************************************************************************
 *
 * #defines local to this file.
 *
 *****************************************************************************/

#define US_PER_LOOP 40000U

#define INITIAL_START_GEAR 1

/*****************************************************************************
 *
 * Constants and variables declared by this file.
 *
 *****************************************************************************/

/* public */

 char  destination_gear_selected;
 char  destination_gear;
 char  flash_desired_allowed;
 char  desired_gear;
 char  desired_gear_dn;          /* debug use - delete later */
 char  desired_gear_up;          /* debug use - delete later */
uchar  coasting_latch;
uchar  sel_gear_cntr1;         /* debug use - delete later */
uchar  sel_gear_cntr2;         /* debug use - delete later */
uchar  sel_gear_cntr3;         /* debug use - delete later */
```

```
uchar  shift_init_type;
uint   lpf_output_speed;
 int   dos_predicted;              /* BIN 0 */
 int   dos_prdtd_lim_no_jake;      /* BIN 0 */


/* local */

/* filter weights for the "mda" output speed filter */
static register uchar w1;
static register uchar w2;
static register uchar w3;
static register uchar w4;

/* shift table (define and extern in shf_tbl.h) */
struct shf_tbl_t shf_tbl;

/* default shift table values */
const struct shf_tbl_t ini_shf_tbl =
{
   1200,                /* aut_dwn_rpm */
   1000,                /* aut_min_dwn_rpm */
   1700,                /* aut_up_rpm */
   0,                   /* best_gr_offset */
   50,                  /* dwn_offset_rpm */
   100,                 /* dwn_reset_rpm */
   400,                 /* dwn_timer_offset_rpm */
   40,                  /* hysteresis_rpm */
   1850,                /* man_dwn_sync_rpm */
   700,                 /* man_up_sync_rpm */
   1900,                /* rated_rpm */
   150,                 /* up_offset_rpm */
   125,                 /* up_reset_rpm */
   200,                 /* up_timer_offset_rpm */
   0,                   /* dwn_accel */
   8,                   /* up_accel */
   3000,                /* offset_time */
   (uint)(0.25*256),    /* aut_up_pct */
   10,                  /* min_output_spd */
   1,                   /* max_start_gear */
   0,                   /* padbyte1 */

   196,                 /* k1_ability, min-ft/rev-sec, BIN 12 */
   431,                 /* axle_ratio_cal, BIN 7 */
   383,                 /* gcw_k1, rev/sec-min-ft, BIN 0 */
   2437,                /* gcw_k2, rev/sec^2, BIN 7 */
   1325,                /* calc_start_point, rpm, BIN 0 */

   107,                 /* k6_ability, min-lb-ft-sec/rev, BIN 8 */
   1500,                /* auto_up_lo_base, rpm, BIN 0 */
   1100,                /* auto_dn_lo_base, rpm, BIN 0 */
   100,                 /* auto_rtd_offset, rpm, BIN 0 */
   1000,                /* lowest_engage_rpm, BIN 0 */
   0,                   /* padword1 */
   0                    /* padword2 */
};


/* local -- initialized at start of task by select_gear */

/* shift points with anti-hunt offsets; referenced by auto_downshift and
   auto_upshift; set by get_automatic_gear and select_gear */
uint upshift_point;
uint downshift_point;

/* lower limit for gear selections */
char lowest_forward;

/* indicate direction of a get_automatic_gear shift; referenced by
   get_manual_gear; cleared by select_gear when shift complete */
char automatic_sip;

/* used in the determination of shift_points based on throttle position */
static uint auto_up_rpm;
static uint auto_dn_rpm;
static uint auto_up_offset_rpm;
static uint auto_dn_offset_rpm;

/* delay counter for anti-hunt */
static uchar antihunt_counter;
```

```
/* gross combined weight calculations */

#define ZERO_SPEED_TIME_LIMIT   4500  /* 3 min @ 0.040 period */
#define VALID_OLD_DATA_TIME      100  /* 4 sec @ 0.040 period */
#define MAX_VEHICLE_WEIGHT    100000  /* 100,000 lbs */
#define DOS_OFFSET_INIT           48
#define ENG_DECEL_LOW_LIMIT     -350  /* rpm/s */
#define ENG_DECEL_LPF            224  /* exp(-2pi(0.53Hz)(0.040s)) = 0.875 (BIN 8) */

#pragma eject
```

```
/*****************************************************************************
 *
 * Function: mda_output_filter
 *
 * Description:
 *      This is a one pole LPF with a variable coefficient.  The magnitude
 *      of the coefficient is directly related to the acceleration content
 *      of the speed sample and the frequency.
 *
 *****************************************************************************/

static void mda_output_filter(void)
{
   #define K1 8
   #define K2 24
   #define K3 48
   #define K4 160

   static register uint os_delta_speed;
   static register uchar weight;

   if (lpf_output_speed > output_speed)
      os_delta_speed = lpf_output_speed - output_speed;
   else
      os_delta_speed = output_speed - lpf_output_speed;

      if (os_delta_speed <= K1)          /* delta <= 200 rpm/s */
      {
         if (w1 > 1) --w1;
         if (w2 < 5) ++w2;
         if (w3 < 6) ++w3;
         if (w4 < 7) ++w4;
         weight = w1;
      }
      else if (os_delta_speed <= K2)   /* 200 rpm/s < delta <= 600 rpm/s */
      {
         if (w1 < 4) ++w1;
         if (w2 > 2) --w2;
         if (w3 < 6) ++w3;
         if (w4 < 7) ++w4;
         weight = w2;
      }
      else if (os_delta_speed <= K3)   /* 600 rpm/s < delta <= 1200 rpm/s */
      {
         if (w1 < 4) ++w1;
         if (w2 < 5) ++w2;
         if (w3 > 3) --w3;
         if (w4 < 7) ++w4;
         weight = w3;
      }
      else if (os_delta_speed <= K4)   /* 1200 rpm/s < delta <= 4000 rpm/s */
      {
         if (w1 < 4) ++w1;
         if (w2 < 5) ++w2;
         if (w3 < 6) ++w3;
         if (w4 > 3) --w4;
         weight = w4;
      }
      else                              /* 4000 rpm/s < delta */
      {
         if (w1 < 4) ++w1;
         if (w2 < 5) ++w2;
         if (w3 < 6) ++w3;
         if (w4 < 7) ++w4;
         weight = 7;
      }

   lpf_output_speed = lpf_output_speed +
      (output_speed >> weight) - (lpf_output_speed >> weight);
}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  new_input_speed
 *
 * Description:
 *      A utility function that returns the input shaft speed expected if
 *      "gear" was engaged with the present output shaft speed.
 *
 *****************************************************************************/

static uint new_input_speed(char gear)
{
   /* new_input_speed = lpf_output_speed * gear_ratio */
   _ax = trn_tbl.gear_ratio[gear + GR_OFS];      /* BIN 8 */
   asm mulu _axbx, lpf_output_speed;             /* BIN 0+8=8 (_axbx) */
   asm shrl _axbx, #8;                           /* BIN 8>>=0 (_ax) */
   return _ax;
}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  auto_upshift
 *
 * Description:
 *      This boolean function returns true when automatic upshift
 *      conditions have been met.
 *
 *****************************************************************************/

static int auto_upshift(void)
{
 /*  if (input_speed > upshift_point) */
   if ((gos > upshift_point) && (output_speed_filtered > 120))
   {
     if (engine_communication_active)
     {
    /*    if ((!shiftability_hold) && (!shiftability_hold_II))  */
           sel_gear_cntr2++;
           return 1;
     }
   }
   return 0;
}


#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  auto_downshift
 *
 * Description:
 *      This boolean function returns true when automatic downshift
 *      conditions have been met.
 *
 *****************************************************************************/

static int auto_downshift(void)
{
 /*  if (input_speed < downshift_point) */
   if (gos < downshift_point)
   {
     return 1;
   }
   return 0;
}
#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  determine_autosplit_type
 *
 * Description:
 *   This function is used to determine if the impending shift type is
 *   MANUAL or AUTO.
 *
 *****************************************************************************/

static char determine_autosplit_type(char passed_new_gear, char passed_initial_gear)
{
   register char new_gr = passed_new_gear;
   register char init_gr = passed_initial_gear;

   if ((shift_in_process == FALSE) || (engine_status == ENGINE_RECOVERY_MODE))
   {
     if ((new_gr ==  1 && init_gr ==  2) ||  /* dn */
         (new_gr ==  3 && init_gr ==  4) ||  /* dn */
         (new_gr ==  5 && init_gr ==  6) ||  /* dn */
         (new_gr ==  7 && init_gr ==  8) ||  /* dn */
         (new_gr ==  9 && init_gr == 10) ||  /* dn */
         (new_gr == 10 && init_gr ==  9) ||  /* up */
         (new_gr ==  8 && init_gr ==  7) ||  /* up */
         (new_gr ==  6 && init_gr ==  5) ||  /* up */
         (new_gr ==  4 && init_gr ==  3) ||  /* up */
         (new_gr ==  2 && init_gr ==  1))    /* up */
       shift_init_type = AUTO;

     else
       shift_init_type = MANUAL;
   }

   if ((init_gr <= 4) && (new_gr < init_gr))
      shift_init_type = MANUAL;   /* prevent coasting auto downshifts in low gears */

}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  get_automatic_gear
 *
 * Description:
 *      This function returns an "automatic" forward gear selection.  It
 *      also performs driver requested shifts (manual_request) restricted
 *      by shaft speeds.  If no gears are available in required direction,
 *      initial_gear is returned.
 *
 *****************************************************************************/

static char get_automatic_gear(char initial_gear, char manual_request)
{
   register char new_gear = initial_gear;

   if (automatic_sip != -1)
   {
       sel_gear_cntr3++;

      /* initiate or continue an automatic upshift: search up from lowest_forward
         (fastest input speed) for the first available gear that will provide input
         speed below a value (approx upshift rpm, minus an offset for gears that will
         result in a net downshift) */
      for (new_gear = lowest_forward;
         (new_input_speed(new_gear) > (upshift_point -
            (new_gear < initial_gear ?
               (shf_tbl.up_offset_rpm + auto_dn_offset_rpm)   : shf_tbl.best_gr_offset)))
         && (new_gear <= trn_tbl.highest_forward);
         ++new_gear)
         ;

      /* if we ran out of gears and the highest is available, it must be due to speed;
         pick highest_forward, input speed will be slower than it is now */
      if (new_gear > trn_tbl.highest_forward)
         new_gear = trn_tbl.highest_forward;

      desired_gear = new_gear;
      desired_gear_up = new_gear;
      determine_autosplit_type(new_gear, initial_gear);

      /* if in gear manual or the selection will underspeed, pick initial_gear */
      if (((shift_init_type == MANUAL) && (transmission_position == IN_GEAR)) ||
          ((automatic_sip == 0) && (new_gear <= initial_gear)))
         new_gear = initial_gear;
      else
      {
         /* indicate gear change and adjust downshift_point */
         automatic_sip = +1;
         auto_up_offset_rpm = 0;
         if (shift_init_type == AUTO)
           auto_dn_offset_rpm = shf_tbl.dwn_timer_offset_rpm;
         else
           auto_dn_offset_rpm = 0;
      }
   }


   if ((automatic_sip != 1) && (initial_gear > lowest_forward))
   {
      /* initiate or continue an automatic downshift:  search down from
         highest_forward (slowest input speed) for the first available gear that will
         provide input speed above a value (approx downshift rpm, plus an offset for
         gears that will result in a net upshift) */
      for (new_gear = trn_tbl.highest_forward;
         (new_input_speed(new_gear) < (downshift_point +
            (new_gear > initial_gear ? shf_tbl.dwn_offset_rpm : shf_tbl.best_gr_offset)))
         && (new_gear >= lowest_forward);
         --new_gear)
         ;

      /* if we ran out of gears and the lowest is available, it must be due to speed;
         pick lowest_forward, input speed will be faster than it is now */
      if (new_gear < lowest_forward)
         new_gear = lowest_forward;

      desired_gear_dn = new_gear;

      if (desired_gear_dn < initial_gear)  /* must be a down shift or else it */
```

```
        desired_gear = new_gear;          /* wrongly cancel the desired_up pick. */

      determine_autosplit_type(new_gear, initial_gear);

      /* if in gear manual or the selection will overspeed, pick initial_gear */
      if (((shift_init_type == MANUAL) && (transmission_position == IN_GEAR)) ||
          ((automatic_sip == 0) && (new_gear >= initial_gear)))
        new_gear = initial_gear;
      else
      {
         /* indicate automatic gear change and adjust upshift_point */
         automatic_sip = -1;
         auto_dn_offset_rpm = 0;
         if (shift_init_type == AUTO)
           auto_up_offset_rpm = shf_tbl.up_timer_offset_rpm;
         else
           auto_up_offset_rpm = 0;
      }
    }
    return new_gear;
}

#if (0)

****  This is the select gear based on AutoShift code ****

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  get_automatic_gear
 *
 * Description:
 *      This function returns an "automatic" forward gear selection.  It
 *      also performs driver requested shifts (manual_request) restricted
 *      by shaft speeds.  If no gears are available in required direction,
 *      initial_gear is returned.
 *
 *****************************************************************************/

static char get_automatic_gear(char initial_gear, char manual_request)
{
   register char new_gear = initial_gear;

   if (((automatic_sip == 0) && auto_upshift()) || (automatic_sip > 0))
   {
       sel_gear_cntr3++;

      /* initiate or continue an automatic upshift: search up from lowest_forward
         (fastest input speed) for the first available gear that will provide input
         speed below a value (approx upshift rpm, minus an offset for gears that will
         result in a net downshift) */
      for (new_gear = lowest_forward;
         (new_input_speed(new_gear) > (upshift_point -
            (new_gear < initial_gear ?
               (shf_tbl.up_offset_rpm + auto_dn_offset_rpm)   : shf_tbl.best_gr_offset)))
         && (new_gear <= trn_tbl.highest_forward);
         ++new_gear)
         ;

      /* if we ran out of gears and the highest is available, it must be due to speed;
         pick highest_forward, input speed will be slower than it is now */
      if (new_gear > trn_tbl.highest_forward)
         new_gear = trn_tbl.highest_forward;

      desired_gear = new_gear;
      determine_autosplit_type(new_gear, initial_gear);

      /* if in gear manual or the selection will underspeed, pick initial_gear */
      if ((shift_init_type == MANUAL) && (transmission_position == IN_GEAR))
         new_gear = initial_gear;
      else
      {
         /* indicate gear change and adjust downshift_point */
         automatic_sip = +1;
         auto_up_offset_rpm = 0;
         if (shift_init_type == AUTO)
           auto_dn_offset_rpm = shf_tbl.dwn_timer_offset_rpm;
         else
           auto_dn_offset_rpm = 0;
      }
    }
    else if (((automatic_sip == 0) &&
              (auto_downshift()) &&
              (initial_gear > lowest_forward)) ||
              (automatic_sip < 0))
    {
       /* initiate or continue an automatic downshift:  search down from
          highest_forward (slowest input speed) for the first available gear that will
          provide input speed above a value (approx downshift rpm, plus an offset for
          gears that will result in a net upshift) */
       for (new_gear = trn_tbl.highest_forward;
          (new_input_speed(new_gear) < (downshift_point +
             (new_gear > initial_gear ? shf_tbl.dwn_offset_rpm : shf_tbl.best_gr_offset)))
          && (new_gear >= lowest_forward);
          --new_gear)
          ;

       /* if we ran out of gears and the lowest is available, it must be due to speed;
          pick lowest_forward, input speed will be faster than it is now */
       if (new_gear < lowest_forward)
          new_gear = lowest_forward;

       desired_gear = new_gear;
       determine_autosplit_type(new_gear, initial_gear);

       /* if in gear manual or the selection will overspeed, pick initial_gear */
```

```
      if ((shift_init_type == MANUAL) && (transmission_position == IN_GEAR))
        new_gear = initial_gear;
      else
      {
         /* indicate automatic gear change and adjust upshift_point */
         automatic_sip = -1;
         auto_dn_offset_rpm = 0;
         if (shift_init_type == AUTO)
           auto_up_offset_rpm = shf_tbl.up_timer_offset_rpm;
         else
           auto_up_offset_rpm = 0;
      }
   }
   return new_gear;
}

#endif

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  determine_destination
 *
 * Description:
 *    This function uses "coasting_latch" to determine if a coasting or
 *    skip shift is being attempted.  When sensed, the latch is used in
 *    the determine_base_pts function to effect the base shift points.
 *
 *****************************************************************************/

void determine_destination(void)
{

   /* if coasting in neutral - force shift points */

   if (coasting_latch == FALSE)
   {
     if ((last_known_gear - destination_gear_selected) > 1) /* multi downshift */
     {
       if ((destination_gear_selected == 7) ||
           (destination_gear_selected == 5) ||
           (destination_gear_selected == 3) ||
           (destination_gear_selected == 1))
       {
         destination_gear_selected++;
         coasting_latch = TRUE;
       }
     }
     else
     {
       if ((destination_gear_selected - last_known_gear) > 1) /* multi upshift */
       {
         if ((destination_gear_selected == 10) ||
             (destination_gear_selected ==  8) ||
             (destination_gear_selected ==  6) ||
             (destination_gear_selected ==  4))
         {
           destination_gear_selected--;
           coasting_latch = TRUE;
         }
       }
     }
   }
   else
    if (shift_in_process == FALSE)
      coasting_latch = FALSE;

}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  determine_manual_shift_pts
 *
 * Description:
 *
 *****************************************************************************/

static void determine_manual_shift_pts(void)
{

   if (coasting_latch == FALSE)
   {
     if ((last_known_gear == 1) ||
         (last_known_gear == 3) ||
         (last_known_gear == 5) ||
         (last_known_gear == 7) ||
         (last_known_gear == 9))
       auto_dn_rpm = 1325;       /* manual downshifts */

     else
       auto_up_rpm = 1375;       /* manual upshifts */
   }

}
#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  determine_base_auto_shift_pts
 *
 * Description:
 *   This function determines the base up and down shift points based on
 *   the position of the throttle.  These base points will be used in the
 *   calculation of the upshift_point and the downshift_point.
 *
 *   The anti-hunting calculations have been moved to this function since
 *   these calculations are now throttle dependent.
 *
 *****************************************************************************/

static void determine_base_auto_shift_pts(void)
{

   if (pct_demand_at_cur_sp > 0)
   {
      /* auto_up_rpm = shf_tbl.auto_up_lo_base +
            ((shf_tbl.aut_up_rpm - shf_tbl.auto_up_lo_base) * %throttle) */

      _cx = shf_tbl.aut_up_rpm - shf_tbl.auto_up_lo_base;
      _bx = pct_demand_at_cur_sp;
      asm mulu _cxdx, _bx;
      asm divu _cxdx, #100;
      auto_up_rpm = shf_tbl.auto_up_lo_base + _cx;

      /* check for RTD requirement */
      if (pct_demand_at_cur_sp > 90)
         auto_up_rpm += shf_tbl.auto_rtd_offset;

      /* auto_dn_rpm = shf_tbl.auto_dn_lo_base +
            ((shf_tbl.aut_dwn_rpm - shf_tbl.auto_dn_lo_base) * %throttle) */

      _cx = shf_tbl.aut_dwn_rpm - shf_tbl.auto_dn_lo_base;
      _bx = pct_demand_at_cur_sp;
      asm mulu _cxdx, _bx;
      asm divu _cxdx, #100;
      auto_dn_rpm = shf_tbl.auto_dn_lo_base + _cx;
   }
   else
   {
      auto_up_rpm = shf_tbl.auto_up_lo_base;
      auto_dn_rpm = shf_tbl.auto_dn_lo_base;
   }

   determine_manual_shift_pts();

   if (shift_in_process)
   {
      /* reset antihunt_counter */
      antihunt_counter = 0;

      /* allow the knob display to flashed any new desired gear */
      flash_desired_allowed = TRUE;
   }
   else
   {
      /* reset shift in process flags and update antihunt_counter */
      automatic_sip = 0;
      if (antihunt_counter < 255)
      {
         ++antihunt_counter;
     /*    flash_desired_allowed = FALSE; */
      }

      /* look for upshift anti-hunt reset conditions */
      if ((antihunt_counter * (US_PER_LOOP/1000)) >= shf_tbl.offset_time)
      {
         /* check for last shift = upshift effects */
         if (auto_dn_offset_rpm == shf_tbl.dwn_timer_offset_rpm)
            auto_dn_offset_rpm = shf_tbl.dwn_offset_rpm;
         else if ((auto_dn_offset_rpm == shf_tbl.dwn_offset_rpm) &&
            (input_speed_filtered > auto_dn_rpm + shf_tbl.dwn_reset_rpm))
            auto_dn_offset_rpm = 0;

         /* check for last shift = downshift effects */
         if (auto_up_offset_rpm == shf_tbl.up_timer_offset_rpm)
```

```
            auto_up_offset_rpm = shf_tbl.up_offset_rpm;
         else if ((auto_up_offset_rpm == shf_tbl.up_offset_rpm) &&
            (input_speed_filtered > auto_up_rpm - shf_tbl.up_reset_rpm))
            auto_up_offset_rpm = 0;

         /* allow the knob display to flashed the desired gear */
         if (((desired_gear > destination_gear_selected) && (gos < (upshift_point + 25))) ||
             ((desired_gear < destination_gear_selected) && (gos > (downshift_point - 25))))
           flash_desired_allowed = FALSE;
         else
           flash_desired_allowed = TRUE;
      }
   }

   /* set the shift points based on throttle and determined offsets */
   upshift_point = auto_up_rpm + auto_up_offset_rpm;
   downshift_point = auto_dn_rpm - auto_dn_offset_rpm;

   /* check that the calculated shift point is reasonable */
   if (upshift_point > shf_tbl.man_dwn_sync_rpm)
      upshift_point = shf_tbl.man_dwn_sync_rpm;

   if (downshift_point < shf_tbl.man_up_sync_rpm)
      downshift_point = shf_tbl.man_up_sync_rpm;

}


#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  select_gear
 *
 * Description:
 *      This is the root function for the periodic task SELECT_GEAR.  Each
 *      loop begins by checking the manual up/down buttons.  Then, based on
 *      selected_mode and output shaft speed, a 'get_..._gear' function is
 *      called to update destination_gear_selected.
 *
 *****************************************************************************/

void select_gear(void)
{
   char manual_request;                /* current manual request (+/- 1) */
   static uchar enable_gcw_calc;       /* diagnostic - delete later !!*/

   enable_gcw_calc = FALSE;            /* diagnostic - delete later !!*/

   shf_tbl = ini_shf_tbl;              /* initialize the shift table */

   destination_gear_selected = 1;
   desired_gear = 1;

   /* initialize file scope variables */

   w1 = 3;
   w2 = 4;
   w3 = 5;
   w4 = 6;
   lpf_output_speed = output_speed;

   upshift_point = shf_tbl.aut_up_rpm;
   downshift_point = shf_tbl.aut_dwn_rpm;
   auto_up_offset_rpm = shf_tbl.up_timer_offset_rpm;
   auto_dn_offset_rpm = shf_tbl.dwn_timer_offset_rpm;
   lowest_forward = INITIAL_START_GEAR;
   automatic_sip = 0;
   antihunt_counter = 255U;
   coasting_latch = FALSE;
   flash_desired_allowed = TRUE;

   x_start_periodic();
   while (1)
   {
      mda_output_filter();      /* update our filtered output speed */
      manual_request = 0;
      determine_base_auto_shift_pts();

      /* set destination_gear_selected from function(s) appropriate for selected_mode */
      switch(selected_mode)
      {
      case REVERSE_MODE:

      case DRIVE_MODE:
         if ((forward_last == TRUE) && (low_speed_latch == FALSE))
         {
           destination_gear_selected = get_automatic_gear(destination_gear_selected, manual_request);
           determine_destination();
           sel_gear_cntr1++;     /* debug use only - delete later */
         }
         break;

      case LOW_MODE:
      case HOLD_MODE:
      case NEUTRAL_MODE:
      case PARK_MODE:
      case POWER_UP_MODE:
      case POWER_DOWN_MODE:
      case DIAGNOSTIC_TEST_MODE:
         /* prevent transient selection upon mode change (these modes ignore it) */
         destination_gear_selected = 0;
         break;

      default:
         /* invalid mode: do nothing */
         break;
      }
      x_sync_periodic(US_PER_LOOP);
```

```
    }
    x_end_periodic();
}
```

```
/*****************************************************************************
 *
 *         Unpublished and confidential.  Not to be reproduced,
 *         disseminated, transferred or used without the prior
 *         written consent of Eaton Corporation.
 *
 *         Copyright Eaton Corporation, 1991-94.
 *         All rights reserved.
 *
 *****************************************************************************
 *
 * Filename:  seq_shft.c96      (AutoSplit)
 *
 * Description:
 *    The functions in this file will perform the required system
 *    level operations for implementing Sequence Shift.
 *
 * Part Number:  <none>
 *
 * $Log:   R:\aselect\vcs\seq_shft.c9v  $
 *
 *    Rev 1.0   12 May 1994 16:26:00   markyvech
 * Initial version
 *
 *****************************************************************************/

/*****************************************************************************
 *
 * Header files included.
 *
 *****************************************************************************/

#include <exec.h>         /* executive information */
#include <c_regs.h>       /* KR internal registers */
#include <wwslib.h>       /* World Wide Software Library */
#include "cont_sys.h"     /* control system information */
#include "conj1939.h"     /* Defines interface to engine communications info */
#include "con_o_s.h"      /* control output signal information */
#include "drl_cmds.h"     /* driveline commands information */
#include "sel_gear.h"
#include "shf_tbl.h"      /* Contains information relative to engine */
#include "trn_tbl.h"      /* transmission table information */
#include "trns_act.h"      /* transmission information */


/*****************************************************************************
 *
 * #defines local to this file.
 *
 *****************************************************************************/

/*****************************************************************************
 *
 * publics.
 *
 *****************************************************************************/

uchar sq_sh1;     /* debug counter - delete later */
uchar sq_sh2;     /* debug counter - delete later */
uchar sq_sh3;     /* debug counter - delete later */


/*****************************************************************************
 *
 * Constants and variables declared by this file.
 *
 *****************************************************************************/

static uchar coast_mode;       /* allows vehicle to coast in low gears */
static uint mode_time_out;     /* time to disengage or synchronize a gear */

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  initialize_sequence_shift
 *
 * Description:
 *   This function initializes the variables to be used in sequence_shift.
 *
 *****************************************************************************/

void initialize_sequence_shift(void)
{
   shift_type = UPSHIFT;
   shift_in_process = FALSE;
}


#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  shift_initiate
 *
 * Description:
 *   This function begins the shift sequence by setting up the
 *   transmission to pull to Neutral, commands the electronic engine
 *   controller to go to zero torque and prepares the clutch to disengage
 *   if required.
 *
 *****************************************************************************/

static void shift_initiate(void)
{

   if (destination_gear < last_known_gear)   /* determine shift type */
   {
     if (pct_demand_at_cur_sp > 5)
       shift_type = POWER_DOWN_SHIFT;
     else
       shift_type = COAST_DOWN_SHIFT;
   }
   else
     shift_type = UPSHIFT;


      /* Attempt to get out of gear for 3 seconds then return fuel to driver */
/*
   if ( (engine_status != ENGINE_PREDIP_MODE) && (mode_time_out > 0) )
      mode_time_out = 300;
   if ( (mode_time_out > 0) && (!coast_mode) )
      --mode_time_out;
*/
   mode_time_out = 300; /* force value for now */

   /* initiate a normal shift sequence */

   /* (do not request engine fueling with engine brake on) */
   eng_brake_command = ENG_BRAKE_OFF;      /* eng brake: zero torque */

   if ((lpf_output_speed < shf_tbl.min_output_spd) ||
       (clutch_state == DISENGAGED) ||
       (mode_time_out == 0) ||
       ((destination_gear < 4) &&
       (coast_mode) &&
       (accelerator_pedal_position <= 5) &&
       (shift_type != UPSHIFT)))
   {
     engine_commands = ENGINE_FOLLOWER;
   }
   else
   {
     engine_commands = ENGINE_PREDIP;      /* engine: bring torque to zero */
     coast_mode = FALSE;
   }
}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  synchronize_gear
 *
 * Description:
 *   This function assists the sychronizing of the transmission if
 *   possible, by controlling input shaft speed through the use
 *   of the clutch, power synchronizer, or inertia brake.   It will
 *   offset sync windows if the shift is taking longer than expected.
 *   It will assist with engagement at rest if the clutch is dragging.
 *
 *****************************************************************************/

static void synchronize_gear(void)
{

   /* turn on engine brakes (J1939) if engine brake assisted shift is requested */

   if (eng_brake_assist)
      eng_brake_command = ENG_BRAKE_FULL;
   else
      eng_brake_command = ENG_BRAKE_OFF;

   /* Attempt to engage for 6 seconds then return fuel control to the driver */
/*
   if ( (engine_status != ENGINE_SYNC_MODE) && (mode_time_out > 0) )
      mode_time_out = 600;
   if ( (mode_time_out > 0) && (!coast_mode) )
      --mode_time_out;
*/
   mode_time_out = 600;  /* force value for now */

   if ((lpf_output_speed < shf_tbl.min_output_spd) ||
       (clutch_state == DISENGAGED) ||
       (mode_time_out == 0) ||
       ((destination_gear < 4) &&
       (coast_mode) &&
       (accelerator_pedal_position <= 5) &&
       (shift_type != UPSHIFT)))
   {
     engine_commands = ENGINE_FOLLOWER;
   }

   else
   {
     engine_commands = ENGINE_SYNC;
     coast_mode = FALSE;
   }
}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  confirm_shift
 *
 * Description:
 *   This function finished the shift; shift_in_process is set FALSE.
 *
 *****************************************************************************/

static void confirm_shift(void)
{
   engine_commands = ENGINE_RECOVERY;        /* engine: finish torque return */
   eng_brake_command = ENG_BRAKE_OFF;        /* eng brake: zero torque */
   mode_time_out = 300;                      /* reset for next shift */
   shift_in_process = FALSE;
   coast_mode = TRUE;
}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  sequence_shift
 *
 * Description:
 *   This function calls the appropriate procedures to perform the
 *   operations of Sequence_Shift depending on the correct state of
 *   the shift process.
 *
 *****************************************************************************/

void sequence_shift(void)
{
   if (destination_gear == NULL_GEAR)  /* system has reset: do not start a shift */
   {
     engine_commands = ENGINE_FOLLOWER;
     eng_brake_command = ENG_BRAKE_IDLE;
   }

   else
     if ((transmission_position == OUT_OF_GEAR) &&
        ((engine_status == ENGINE_SYNC_MODE) ||
         (engine_status == ENGINE_PREDIP_MODE))) /* forces call shift_initiate() */
     {
       sq_sh1++;
       synchronize_gear();
     }

     else
       if (((engine_status == ENGINE_SYNC_MODE) ||
            (engine_status == ENGINE_RECOVERY_MODE)) &&
            (destination_gear == current_gear) &&
            (transmission_position == IN_GEAR))
       {
         sq_sh2++;
         confirm_shift();
       }

       else
         if (((destination_gear != current_gear) &&      /* auto splitter */
              (low_speed_latch == FALSE) &&
              (automatic_sip != 0) &&
              (transmission_position == IN_GEAR)) ||
             ((transmission_position == OUT_OF_GEAR) &&  /* manual shift */
              (low_speed_latch == FALSE)))
         {
           shift_initiate();
           sq_sh3++;
         }
}
```

```
/*****************************************************************************
 *
 *        Unpublished and confidential.  Not to be reproduced,
 *        disseminated, transferred or used without the prior
 *        written consent of Eaton Corporation.
 *
 *        Copyright Eaton Corporation, 1994.
 *        All rights reserved.
 *
 *****************************************************************************
 *
 * Filename:  trns_act.c96    (AutoSplit)
 *
 * Description:
 *  This modules monitors and controls the transmission actions.
 *
 * Part Number:  <none>
 *
 * $Log:   ??? $
 *
 *    Rev 1.0   3 May 1994 13:35:04   markyvech
 * Initial revision.
 *
 *****************************************************************************/
/*****************************************************************************
*
* Header files included.
*
*****************************************************************************/

#include <exec.h>       /* executive information */
#include <c_regs.h>        /* KR internal register definitions          */
#include <kr_sfr.h>     /* defines the kr special function registers */
#include <kr_def.h>     /* 80c196kr bits, constants, and structures */
#include <wwslib.h>
#include "drl_cmds.h"   /* engine control interface */
#include "trns_act.h"   /* interface to this module */
#include "trn_tbl.h"    /* transmission table */
#include "calc_spd.h"
#include "cont_sys.h"
#include "sel_gear.h"
#include "conj1939.h"

/*****************************************************************************
*
*  Variables declared by this file.
*
*****************************************************************************/

register unsigned char   transmission_position;

unsigned char   low_speed_latch;
unsigned char   forward_last;
unsigned char   splitter_hi;
unsigned char   splitter_lo;
unsigned char   splitter_timer;
unsigned char   splitter_within_sync;
unsigned char   aux_box;
  signed char   g_ptr_old;
  signed char   current_gear;
  signed char   last_known_gear;
unsigned int    gear_in_timer;
unsigned int    gear_out_timer;
unsigned int    abs_trans_sync_error;
unsigned int    trans_window_calc;
  signed int    input_speed_modified;
  signed int    trans_sync_error;
  signed int    range_error;
  signed int    range_cal;
  signed int    splitter_tc;

  signed long   isdgf;
  signed long   gros;
  signed char   g_ptr;

#pragma eject
```

```
/*****************************************************************************
*
* #defines and constants local to this file.
*
*****************************************************************************/

#define US_PER_LOOP 10000U
#define RKM_US_PER_LOOP 40000U

static const uchar SPLITTER_LO_TABLE[23] =
{
      0,      /* -4 */
      0,      /* -3 */
     ON,      /* -2 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
     ON,      /* -1 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /*  0 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /*  1 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /*  2 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
     ON,      /*  3 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /*  4 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
     ON,      /*  5 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /*  6 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
     ON,      /*  7 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /*  8 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
     ON,      /*  9 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /* 10 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
      0,      /* 11 */
      0,      /* 12 */
      0,      /* 13 */
      0,      /* 14 */
      0,      /* 15 */
      0,      /* 16 */
      0,      /* 17 */
      0       /* 18 */
};

static const uchar SPLITTER_HI_TABLE[23] =
{
      0,      /* -4 */
      0,      /* -3 */
     ON,      /* -2 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
    OFF,      /* -1 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
    OFF,      /*  0 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
    OFF,      /*  1 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /*  2 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
    OFF,      /*  3 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /*  4 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
    OFF,      /*  5 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /*  6 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
    OFF,      /*  7 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /*  8 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
    OFF,      /*  9 */  /* split_lo =  ON,  split_hi = OFF;  direct    */
     ON,      /* 10 */  /* split_lo = OFF,  split_hi =  ON;  overdrive */
      0,      /* 11 */
      0,      /* 12 */
      0,      /* 13 */
      0,      /* 14 */
      0,      /* 15 */
      0,      /* 16 */
      0,      /* 17 */
      0       /* 18 */
};

#pragma eject
```

```
static const uchar SPLITTER_TC_TABLE[23] =
{                       /* Splitter movement time constant in milliseconds */
      0,        /* -4 */
      0,        /* -3 */
    100,        /* -2 */  /* splitter = overdrive */
    100,        /* -1 */  /* splitter = direct    */
    100,        /*  0 */  /* splitter = direct    */
    100,        /*  1 */  /* splitter = direct    */
    100,        /*  2 */  /* splitter = overdrive */
    100,        /*  3 */  /* splitter = direct    */
    100,        /*  4 */  /* splitter = overdrive */
    100,        /*  5 */  /* splitter = direct    */
    100,        /*  6 */  /* splitter = overdrive */
    100,        /*  7 */  /* splitter = direct    */
    100,        /*  8 */  /* splitter = overdrive */
    100,        /*  9 */  /* splitter = direct    */
    100,        /* 10 */  /* splitter = overdrive */
      0,        /* 11 */
      0,        /* 12 */
      0,        /* 13 */
      0,        /* 14 */
      0,        /* 15 */
      0,        /* 16 */
      0,        /* 17 */
      0         /* 18 */
};

#pragma eject
```

```
/*****************************************************************************
*
* Function:  Initialize_Trans_Action
*
* Description:
*   This function initializes those module variables that must be set to a
*   know state on power up or reset.
*
*****************************************************************************/

void initialize_trans_action(void)
{

  gear_in_timer = 200;
  gear_out_timer = 200;
  g_ptr_old = 0;
  current_gear = 0;
  last_known_gear = 0;
  destination_gear = 0;
  transmission_position = OUT_OF_GEAR;
  low_speed_latch = TRUE;
  splitter_lo = 0;
  splitter_hi = 0;

}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  determine_gear
 *
 * Description:
 *  This function determines the current gear that the transmission
 *  is in.  When conditions are such that the current gear can not be
 *  determined it will be set to a default,(0).
 *
 *  Note: When the error across the transmission is near zero for some
 *  time for a given test gear then it will be deemed in that gear.
 *
 *          error = input_spd/gf[gear] - gr[gear] * os
 *
 *****************************************************************************/
void determine_gear(void)
{

#define BIN_8              256      /*    2 BIN 8  */
#define MAX_ERR           4000      /*      RPM    */
#define WINDOW              30      /*   30 RPM    */
#define GEAR_IN_TIME_LEVER  30      /*  300 MSEC   */
#define GEAR_IN_TIME_AUTO   20      /*  200 MSEC   */
#define GEAR_OUT_TIME        8      /*   80 MSEC   */
#define ERROR_FUDGE_FACTOR   3      /*      RPM    */
/*
signed long   isdgf;
signed long   gros;
signed char   g_ptr;
*/


#if (0)
   g_ptr = -1; /* lowest reverse ratio */

/*   isdgf = (((signed long) input_speed_filtered) * BIN_8) / trn_tbl.GF[g_ptr + GR_OFS]; */
   _bx = (signed int)(input_speed_filtered);
   _cx = BIN_8;
   _ax = trn_tbl.GF[g_ptr + GR_OFS];
   asm mul _cxdx, _bx;
   asm div _cxdx, _ax;
   isdgf = _cx;

/*   gros  = (((signed long) output_speed_filtered) * trn_tbl.GR[g_ptr + GR_OFS]) / BIN_12; */
   _bx = (signed int)(output_speed_filtered + ERROR_FUDGE_FACTOR);
   _cx = trn_tbl.GR[g_ptr + GR_OFS];
   _ax = BIN_8;
   asm mul _cxdx, _bx;
   asm div _cxdx, _ax;
   gros = _cx;

   trans_sync_error = (isdgf - gros);
   if (isdgf > gros)
      abs_trans_sync_error = (unsigned int)(isdgf - gros);
   else
      abs_trans_sync_error = (unsigned int)(gros - isdgf);
#endif

   abs_trans_sync_error = MAX_ERR;
   trans_window_calc = 0;

   if (abs_trans_sync_error > trans_window_calc)  /* if not in reverse, check for forward */
   {
     g_ptr = 1 + trn_tbl.highest_forward;
     abs_trans_sync_error = MAX_ERR;
     while ((abs_trans_sync_error > trans_window_calc) && (g_ptr != 0))
       {
         g_ptr--;
     /*  isdgf = (((signed long) input_speed_filtered) * BIN_8) / trn_tbl.GF[g_ptr + GR_OFS]; */
         _bx = (signed int)(input_speed_filtered);
         _cx = BIN_8;
         _ax = trn_tbl.GF[g_ptr + GR_OFS];
         asm mul _cxdx, _bx;
         asm div _cxdx, _ax;
         isdgf = _cx;

     /*  gros  = (((signed long) output_speed_filtered) * trn_tbl.GR[g_ptr + GR_OFS]) / BIN_12; */
         _bx = (signed int)(output_speed_filtered + ERROR_FUDGE_FACTOR);
         _cx = trn_tbl.GR[g_ptr + GR_OFS];
```

```
      _ax = BIN_8;
      asm mul _cxdx, _bx;
      asm div _cxdx, _ax;
      gros = _cx;

      trans_sync_error = isdgf - gros;
      if (isdgf > gros)
        abs_trans_sync_error = (int)(isdgf - gros);
      else
        abs_trans_sync_error = (int)(gros - isdgf);

     /* calculate trans sync error window based on gear pointer */
      _bx = WINDOW;                          /* BIN 0 */
      _cx = BIN_8;                           /* BIN 8 */
      _ax = trn_tbl.GF[g_ptr + GR_OFS];      /* BIN 8 */
      asm mulu _cxdx, _bx;                   /* make WINDOW BIN 8 */
      asm divu _cxdx, _ax;                   /* divide by front ration BIN 8 */
      trans_window_calc = _cx;               /* BIN 0 */
    }
}

if (g_ptr == 0)                              /* If in neutral, force values */
{
  abs_trans_sync_error = MAX_ERR;
  trans_sync_error = MAX_ERR;
  trans_window_calc = 0;
  isdgf = 0;
  gros = 0;
}

if ((abs_trans_sync_error > trans_window_calc) ||     /* Must have error for some */
  ((g_ptr != current_gear) && (current_gear != 0)))  /* before neutral state is  */
{                                                    /* recognized.              */
  if (gear_out_timer == 0)
  {
    transmission_position = OUT_OF_GEAR;
    current_gear = 0;
  }

  else
    gear_out_timer--;
}
else
  gear_out_timer = GEAR_OUT_TIME;


if ((g_ptr != g_ptr_old) || (g_ptr == 0) ||   /* if not in gear, init gear in timer.       */
    ((accelerator_pedal_position < 5) &&      /* Rule out picking a gear when coasting     */
     (input_speed < 800) &&                   /* down in neutral and no throttle.          */
     (low_speed_latch == FALSE)))             /* (Found that idle speed and output speed   */
{                                             /* would natch a gear even when in neutral.) */
  if ((engine_commands == ENGINE_SYNC) ||
      (engine_commands == ENGINE_PREDIP))
  {
     if (shift_init_type == AUTO)
       gear_in_timer = GEAR_IN_TIME_AUTO;
     else
       gear_in_timer = GEAR_IN_TIME_LEVER;
  }

}

else
  if (gear_in_timer == 0)
  {
    current_gear = g_ptr;
    last_known_gear = g_ptr;
    transmission_position = IN_GEAR;

    if ((gos_current_gear > (downshift_point + 100)) &&
        (low_speed_latch == TRUE))
    {
      low_speed_latch = FALSE;
      destination_gear = current_gear;
      destination_gear_selected = current_gear;
      desired_gear = current_gear;
      lowest_forward = current_gear;
    }
    else
    {
```

```
        if (low_speed_latch == TRUE)
        {
          destination_gear = lowest_forward;          /* was set to 1 */
          destination_gear_selected = lowest_forward; /* was set to 1 */
          desired_gear = lowest_forward;              /* was set to 1 */
          shift_in_process = FALSE;
        }
      }

      if (last_known_gear > 0)              /* Record REV/FOR data for */
        forward_last = TRUE;                /* use in the select_gear  */
      else                                  /* module.                 */
        forward_last = FALSE;
    }
    else
      gear_in_timer--;

  g_ptr_old = g_ptr ;

  if (output_speed_filtered < 80) /* If stopped - current_gear = first. */
  {
    current_gear = 0 ;
    transmission_position = OUT_OF_GEAR;
    low_speed_latch = TRUE;
  }
}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  determine_range_status
 *
 * Description:
 *  This function determines the status the of range.
 *
 *  rng_err = rear_counter_spd - (range_ratio * output_spd)
 *
 *  rcs = 54/21 * 44 * os (for low range)
 *
 *  rcs = 42/51 * 44 * os (for high range)
 *
 *****************************************************************************/

void determine_range_status(void)
{

#define BIN_12             4096   /* 2 bin 12   */
#define HI_RANGE_GEAR         7
#define LO_RANGE_CAL      10532   /* 54/21 BIN 12 */
#define HI_RANGE_CAL       3373   /* 42/51 BIN 12 */
#define RANGE_WINDOW_POS     30   /*  30 RPM      */
#define RANGE_WINDOW_NEG    -30   /* -30 RPM      */


  if (destination_gear >= HI_RANGE_GEAR)
    range_cal = HI_RANGE_CAL;
  else
    range_cal = LO_RANGE_CAL;

  range_error =(((aux_speed * BIN_12)
               - (range_cal * output_speed_filtered))/BIN_12);

  if ((range_error > RANGE_WINDOW_POS) || (range_error < RANGE_WINDOW_NEG))
    aux_box = OUT_OF_GEAR;
  else
    aux_box = IN_GEAR;

}

#pragma eject
```

```
/*****************************************************************************
 *
 * Function:  determine_splitter
 *
 * Description:
 *  This function determines the correct state for the splitter.
 *  Once the transmission is in gear both splitter solenoids are turned off.
 *
 *****************************************************************************/

void determine_splitter_state(void)
{
#define SPLTR_SYNC_OFFSET_POS    80   /*  80 RPM   */
#define SPLTR_SYNC_OFFSET_NEG   -80   /* -80 RPM   */
#define SPLITTER_TIME            20   /* 200 MSEC  */


  if (engine_status == ENGINE_PREDIP_MODE)
    splitter_timer = SPLITTER_TIME;
  else
    if (splitter_timer > 0)
      splitter_timer--;


  splitter_tc = SPLITTER_TC_TABLE[destination_gear + GR_OFS];

  input_speed_modified = (signed int)(input_speed_filtered) +
                         (input_speed_accel_filtered/(1000/splitter_tc));

  if ((input_speed_modified < (gos_signed + SPLTR_SYNC_OFFSET_POS)) &&
      (input_speed_modified > (gos_signed + SPLTR_SYNC_OFFSET_NEG)))
    splitter_within_sync = TRUE;
  else
    splitter_within_sync = FALSE;


  if ((splitter_timer > 0) ||

      ((transmission_position == IN_GEAR) &&       /* debug - delete later */
       (shift_in_process == FALSE))           ||  /* debug - delete later */

      (low_speed_latch == TRUE) ||

      (engine_status == ENGINE_RECOVERY_MODE) ||

      ((shift_init_type == MANUAL) &&
       (engine_status == ENGINE_SYNC_MODE)) ||

      ((shift_init_type == AUTO) &&
       (engine_status == ENGINE_SYNC_MODE) &&
       (splitter_within_sync == TRUE)))
  {
    splitter_hi = SPLITTER_HI_TABLE[destination_gear + GR_OFS];
    splitter_lo = SPLITTER_LO_TABLE[destination_gear + GR_OFS];
  }

  else
  {
    splitter_hi = OFF;
    splitter_lo = OFF;
  }
}

#pragma eject
```

```
/*****************************************************************************
*
* Function:  Transmission_Action
*
* Description:
*      This funtion controls the states of the system ouput devices.
*
*****************************************************************************/
void transmission_action(void)
{

  initialize_trans_action();      /* initialize variables */

   x_start_periodic();
   while (1)
   {

    determine_gear();             /* calculate the current gear */
    determine_range_status();     /* determine range state */
    determine_splitter_state();   /* determine correct state for splitter */

    x_sync_periodic(US_PER_LOOP);
   }
   x_end_periodic();
}
```